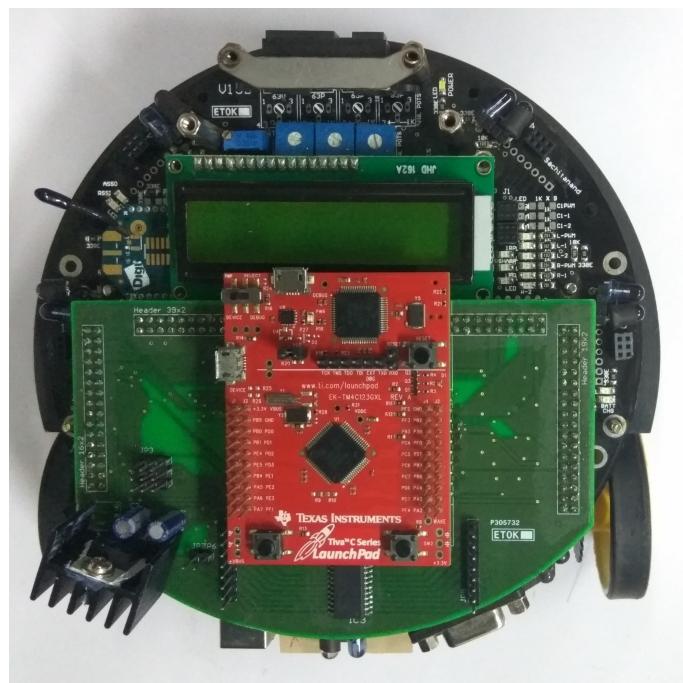


Tiva Based Daughter Board for Firebird V Hardware And Software Manual.

eRTS Lab IIT Bombay

July 4, 2017



1 Credits

Version 1.0
July 4, 2017

Documentation Author(Alphabetical Order):

1. Ayush Gaurav, Intern eYSIP 2017
2. Nagesh K, Intern eYSIP 2017

Credits(Alphabetical Order):

1. Prof Kavi Arya, CSE IIT Bombay
2. Nex Robotics Pvt. Ltd.
3. Piyush Manavar, Team e-Yantra
4. Saurav Shandilya, Team e-Yantra

2 Notice

The contents of this manual are subject to change without notice. All efforts have been made to ensure the accuracy of contents in this manual. However, should any errors be detected, e-yantra welcomes your corrections. You can send us your queries / suggestions at [Contact Us](#)

Table Of Content

1 Credits	2
2 Notice	3
3 Introduction	7
3.1 Safety precautions:	7
3.2 Inappropriate Operation:	7
4 Tiva Based Daughter Board	8
4.1 Technical Specification	
.	8
5 Hardware Manual:	9
5.1 Voltage Regulation on the Daughter Board	9
5.1.1 Powering Micro-controller	9
5.1.2 Powering Servos	9
5.2 Level Converters	10
5.3 Sensors	10
5.3.1 3.3V sensors	11
5.3.2 5V sensors	11
5.4 Port Expander	12
5.5 External ADC	13
5.6 LCD Interfacing	13
5.7 USB Communication	15
5.8 Reset Switch	15
5.9 Servo Connectors	16
5.10 TM4C123GH6PM Micro-controller:	17
5.10.1 Feature of TM4C123GH6PM	17
5.11 Pin Functionality	19
5.11.1 Pin of uC	19

5.11.2	Robot Main Board Connections	20
5.11.3	Pin Connection Of Plug And Play Board	22
6	Software Manual:	24
6.1	Code Composer Studio:	24
6.1.1	Download CC Studio:	24
6.1.2	Installing C C Studio:	24
6.1.3	Create a New Project	26
6.1.4	Add Path and Build Variables	26
6.2	Buzzer	32
6.2.1	Code	32
6.3	Simple I/O Operation	33
6.3.1	Code for Plug and Play Board	33
6.4	Robot Direction Control	34
6.4.1	Code for Plug and Play Board:	35
6.4.2	Code for uC based Board:	36
6.5	Robot Position Control Using Interrupts	37
6.5.1	Code for Plug and Play Board:	37
6.5.2	Code for uC based Board:	40
6.6	Timers and its Interrupts	42
6.6.1	Code	43
6.7	Robot Speed Control	44
6.7.1	Pulse Width Modulation(PWM)	44
6.7.2	Code for Plug and Play Board:	45
6.7.3	Code for uC based Board:	47
6.8	LCD Interfacing	49
6.8.1	Code for Plug and Play Board:	49
6.8.2	Code for uC based Board	52
6.9	Analog To Digital Converter	54
6.9.1	Code for Plug and Play Board:	55
6.9.2	Code for uC based Board	57
6.10	External ADC Interface	59

6.10.1	Code for Plug and Play Board:	59
6.10.2	Code for uC based Board:	63
6.11	Serial Communication	66
6.11.1	Connections	66
6.11.2	Code for Plug and Play Board:	67
6.11.3	Code for uC based Board:	68
6.12	I2C Communication	69
6.12.1	Introduction	69
6.12.2	Features	69
6.12.3	Working	69
6.12.4	Connections	70
6.12.5	Code for Plug and Play Board:	70
6.12.6	Code for uC based Board:	73

3 Introduction

Tiva Daughter board for Fire Bird V will help you gain exposure to the world of robotics and embedded systems with ARM Cortex M4. The board is designed with Open Source Philosophy in software and hardware design ,you will be able to create and contribute to complex applications that run on this platform, helping you acquire expertise as you spend more time with them.

3.1 Safety precautions:

- Robot's electronics is static sensitive. Use robot in static free environment.
- Read the assembling and operating instructions before working with the robot.
- If robot's battery low buzzer starts beeping, immediately charge the batteries.
- To prevent fire hazard, do not expose the equipment to rain or moisture.
- Refrain from dismantling the unit or any of its accessories once robot is assembled.
- Charge the NiMH battery only with the charger provided on the robot.
- Never allow NiMH battery to deep discharge.
- Mount all the components with correct polarity.
- Keep wheels away from long hair or fur.
- Keep the robot away from the wet areas. Contact with water will damage the robot.
- To avoid risk of fall, keep your robot in a stable position.
- Do not attach any connectors while robot is powered ON.
- Never leave the robot powered ON when it is not in use.
- Disconnect the battery charger after charging the robot.

3.2 Inappropriate Operation:

Inappropriate operation can damage your robot. Inappropriate operation includes, but is not limited to:

- Dropping the robot, running it off an edge, or otherwise operating it in irresponsible manner.
- Interfacing new hardware without considering compatibility.
- Overloading the robot above its payload capacity.
- Exposing the robot to wet environments.
- Continuing to run the robot after hair, yarn, string, or any other item is entangled in the robot's axles or wheels.
- All other forms of inappropriate operations.
- Using robot in areas prone to static electricity.
- Read carefully paragraphs marked with caution symbol.

4 Tiva Based Daughter Board

There are two daughter boards one with the launchpad and other one with the Arm Cortex M4 based uC. Almost all the specification are same unless mentioned otherwise.

4.1 Technical Specification

Microcontroller:

TM4C123gh6pm (ARM architecture based Microcontroller)

To know more about the microcontroller please refer to [datasheet](#).

Sensors:

Three white line sensors (extendable to 7)

Five Sharp GP2Y0A02YK IR range sensor (One in default configuration)

Eight analog IR proximity sensors

Two position encoders

Indicators:

2 x 16 Characters LCD

Buzzer

Communication:

USB Communication

Wireless ZigBee Communication (2.4GHZ) (if XBee wireless module is installed)

Bluetooth communication(Can be interfaced on external UART0 available on the board)

Simplex infrared communication (From infrared remote to robot)

I2C Communication

Battery Life:

2 Hours, while motors are operational at 75% of time

Locomotion:

Two DC geared motors in differential drive configuration and caster wheel at front as support

Top Speed: 24 cm / second

Wheel Diameter: 51mm

Position encoder: 30 pulses per revolution

Position encoder resolution: 5.44 mm

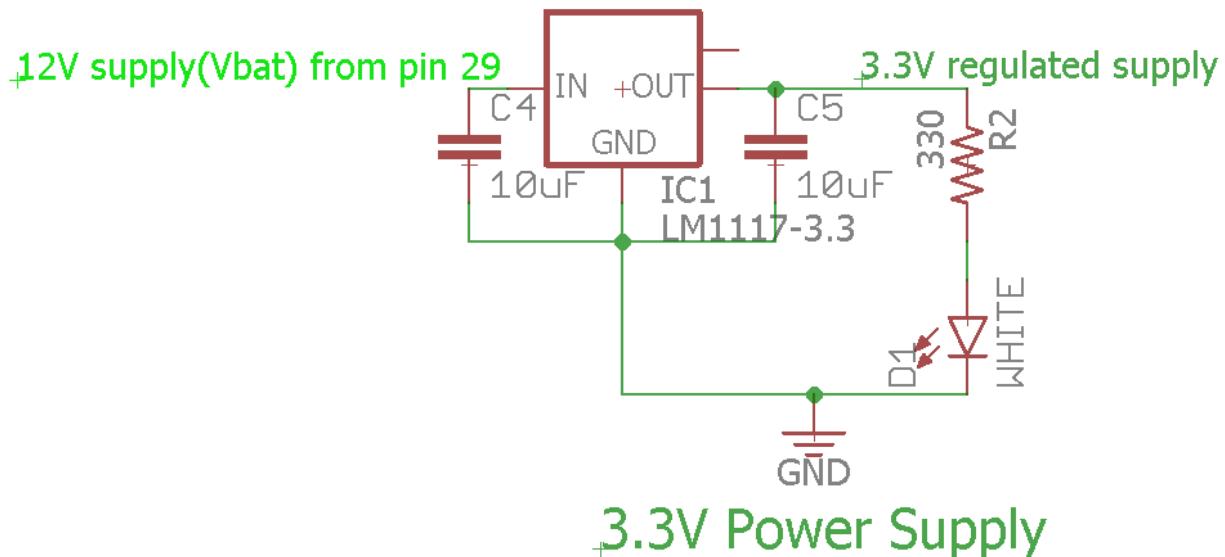
5 Hardware Manual:

5.1 Voltage Regulation on the Daughter Board

The voltage source available on the Firebird is 9.6V. But the TIVA platform works on 3.3V and the servos can operate upto 6V. So there must be 3 different voltage levels on the board. The uC based board has 2 voltage regulators and the plug and play board has 1 voltage regulator. In the uC based board the 9.6 volts is 3.3V to power the microcontroller. In the plug and play board the there is an inbuilt voltage regulator, so it is directly connected connected to 5v, 300mA source. The servo in both the boards has a separate 5V regulator.

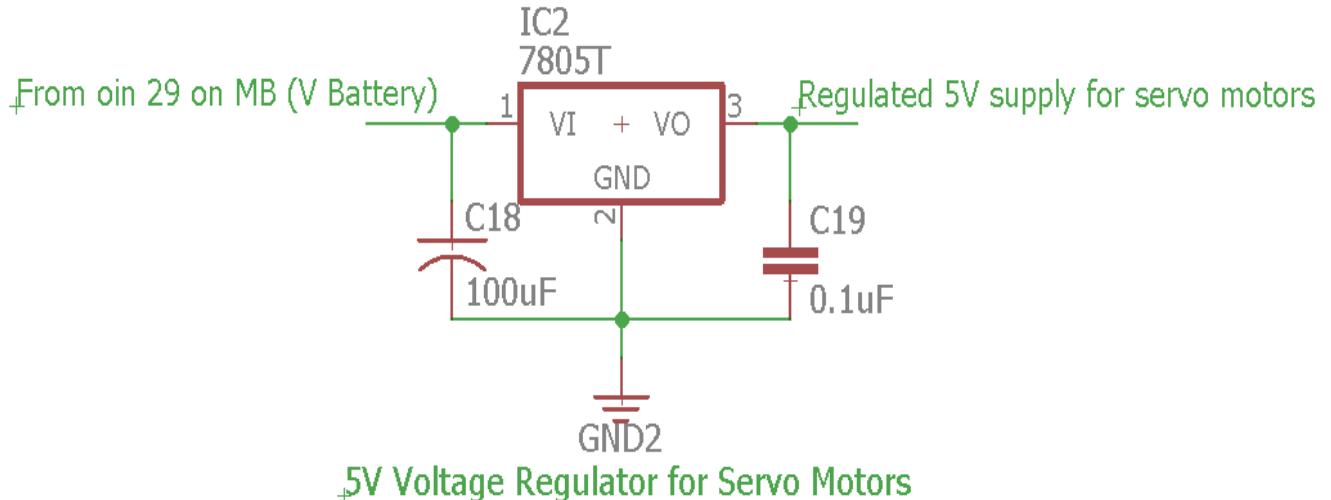
5.1.1 Powering Micro-controller

The boards have different powering circuits. In the plug and play board is connected to 5V source on Pin 10. In the uC based board the 9.6V source available on Pin 29 is reduced to 3.3V. Refer to the schematic below for further details.



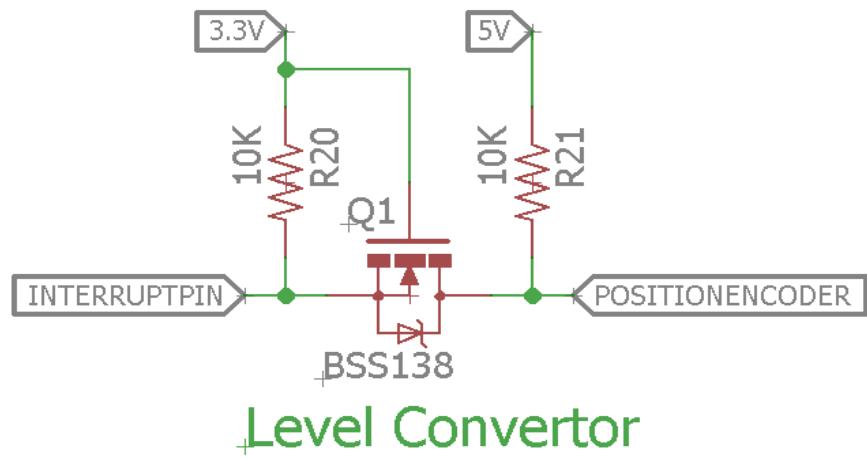
5.1.2 Powering Servos

The servo motors can operate safely up to 6V, beyond this voltage they get damaged. Also, the servos require high current. There is a separate power line for servos taken from Pin 29 and reduced to 5V using the voltage regulator. Refer the schematic for further details.



5.2 Level Converters

The TIVA platform operates at 3.3V and the Firebird operates at 5V. Directly connecting these pins to the TIVA may be fatal. So to interface these sensors, a level converter is used. A bidirectional MOS-FET based level converter used. The level converter is necessary for input pins. In the boards Level converter is used for interfacing the position encoders of the motors. Refer the schematic for further details.



NOTE: If the user wishes to interface extra sensors using the GPIOs provided on the board, then external level converters have to be used if the output of the sensor is above 3.3V.

5.3 Sensors

The firebird V has as many as 22 sensors, but maximum 12 sensors can be interfaced directly with the controller. The daughter board has interfaced 20 of those 22 sensors using external I2C bases ADC. Sensors that were not included in the daughter board are current sensor and battery monitoring sensor. These sensors are working either on 3.3V or on 5V. Interfacing 3.3V sensors are simple and can be directly connected to the controller. On the other hand 5V can not be directly interfaced so a different approach is taken which will be mentioned in the 5V sensors sub heading.

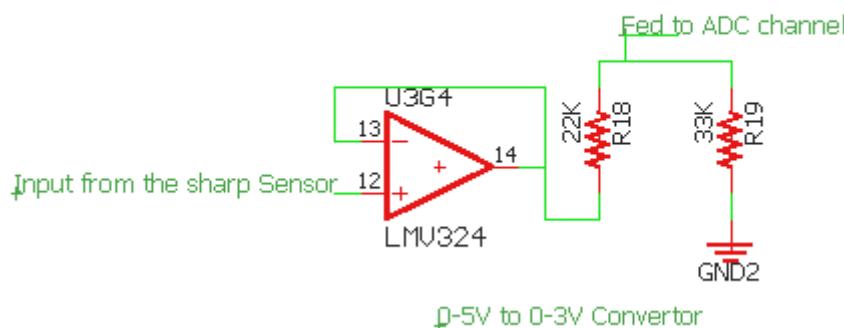
5.3.1 3.3V sensors

The output white line sensors and IR Proximity sensors vary from 0 to 3.3V. Hence these sensors can be interfaced directly with the microcontroller. Refer the table below for pin connections.

IR Proximity Sensors	Pin Name(uC)	Pin Name(Plug and Play)
1 PE1	PB5	
2 PE3	PD0	
3 PE5	PD3	
4 PE4	PD1	
5 PB5	PE5	
6 External ADC IN6	External ADC IN7	
7 External ADC IN7	PE0	
8 External ADC IN0	External ADC IN0	
White Line Sensors	Pin Name(uC)	Pin Name(Plug and Play)
1 PD2	PE1	
2 PD1	PE2	
3 PD0	PE3	
4 External ADC IN1	External ADC IN2	
5 External ADC IN2	External ADC IN3	
6 External ADC IN3	External ADC IN4	
7 External ADC IN4	External ADC IN5	

5.3.2 5V sensors

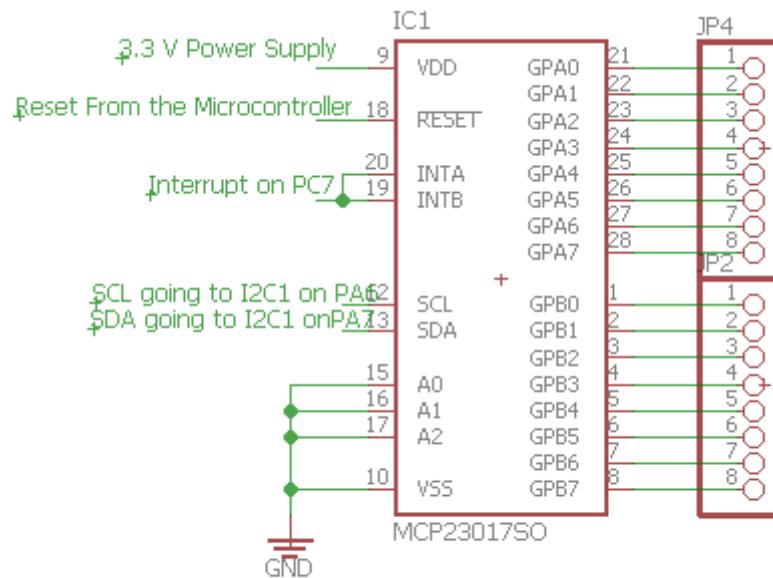
Sharp Sensors are the only sensors on board that works on 5V supply. The output of the sharp sensor ranges from 0-5V and according to the output we have a formula to calculate the distance. While uC has VREF as 3.3V so these sensors cannot be directly connected. The approach we followed is to feed the output of the sensor to a buffer and then using a voltage divider convert 0-5 range to 0-3V range. For better understanding refer to the schematic below. There is also a table which tells about the pin connection.



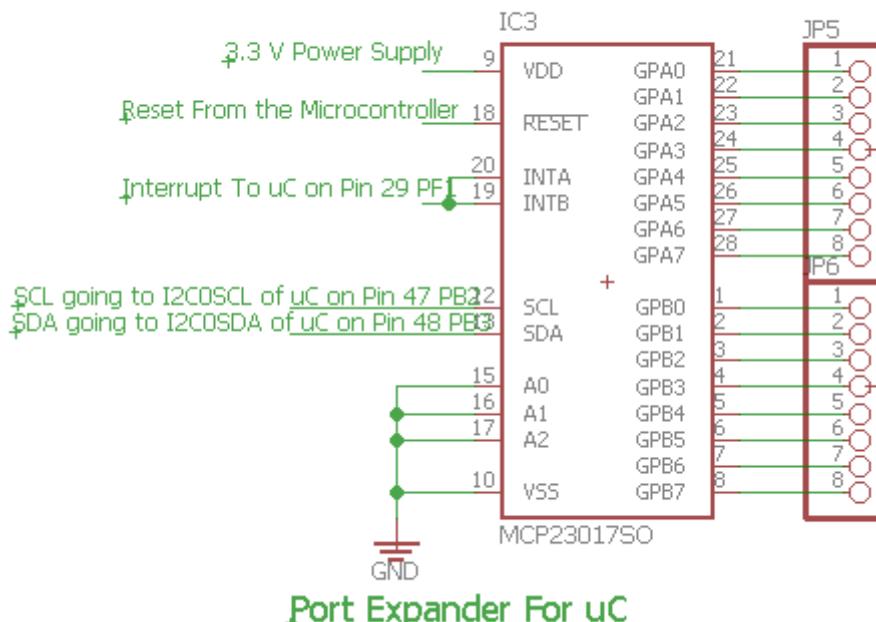
Sharp Sensors	Pin Name(uC)	Pin Name(Plug and Play)
1 PE0	PB4	
2 PE2	External ADC IN1	
3 PD3	PD2	
4 External ADC IN5	External ADC IN6	
5 PB4	PE4	

5.4 Port Expander

TM4C123GH6PM has only 64 pins out of which only 43 are GPIO pins. This limits our application to read input and respond correspondingly. To increase the number of GPIO and there interrupts we have used I2C compatible a port expander MCP23017. It has 2 PORTS A and B, with each port having 8 Pins. The interrupts on each pin can also be monitored. To read more about it, download the datasheet from [here](#). The schematic of the connection is shown below. Keep in mind that I2C SCL and SDA have already been pulled up using 10K resistor.



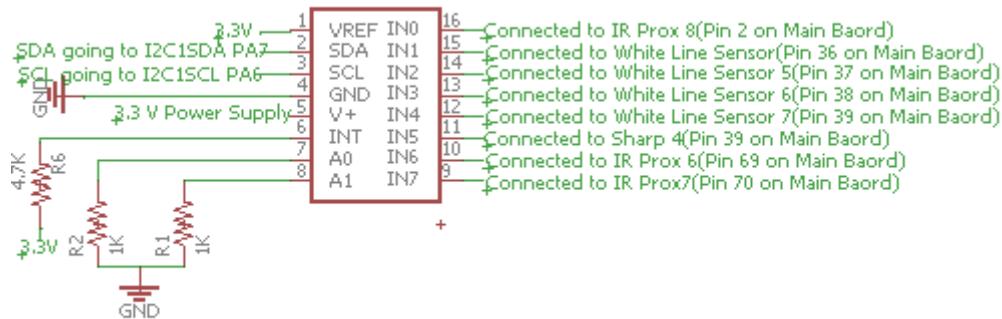
Port Expander For Plug and Play



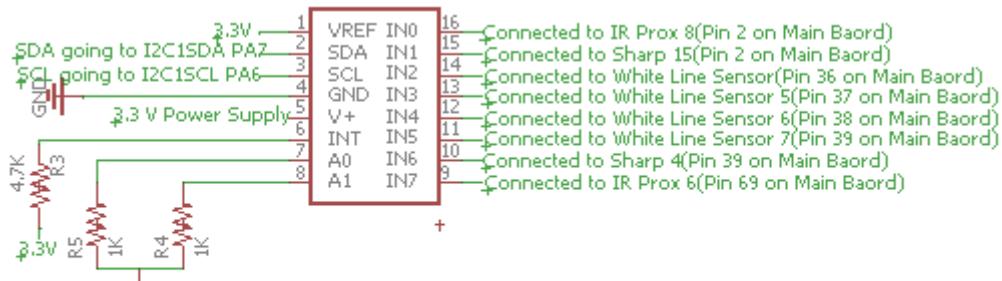
Port Expander For uC

5.5 External ADC

It has already been mentioned that adc channels on the microcontroller is limited to 12 while firebird has 22 sensors available. We have interfaced an external ADC which is also I2C compatible. It has 8 channel with 12 bit resolution. To read more about it, download the datasheet from [here](#). The schematic of the connection is shown below. Keep in mind that I2C SCL and SDA have already been pulled up using 10K resistor.



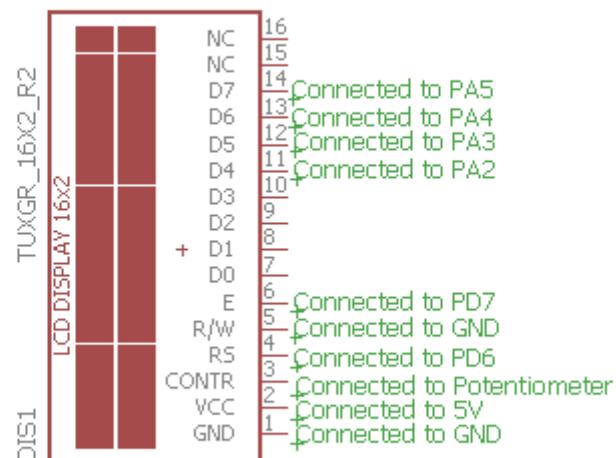
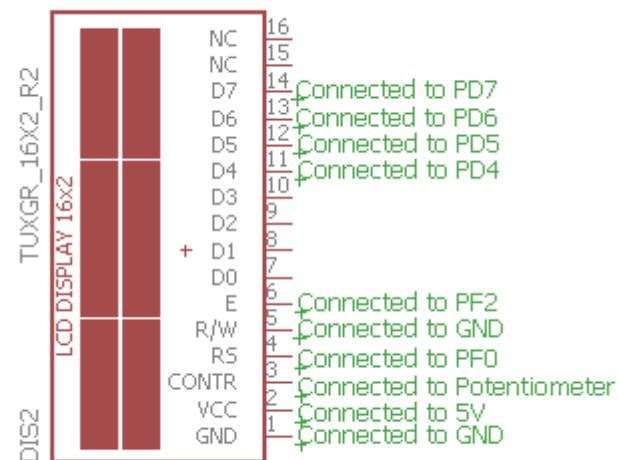
External ADC connection for uC Board



External ADC connections for Plug and Play Board

5.6 LCD Interfacing

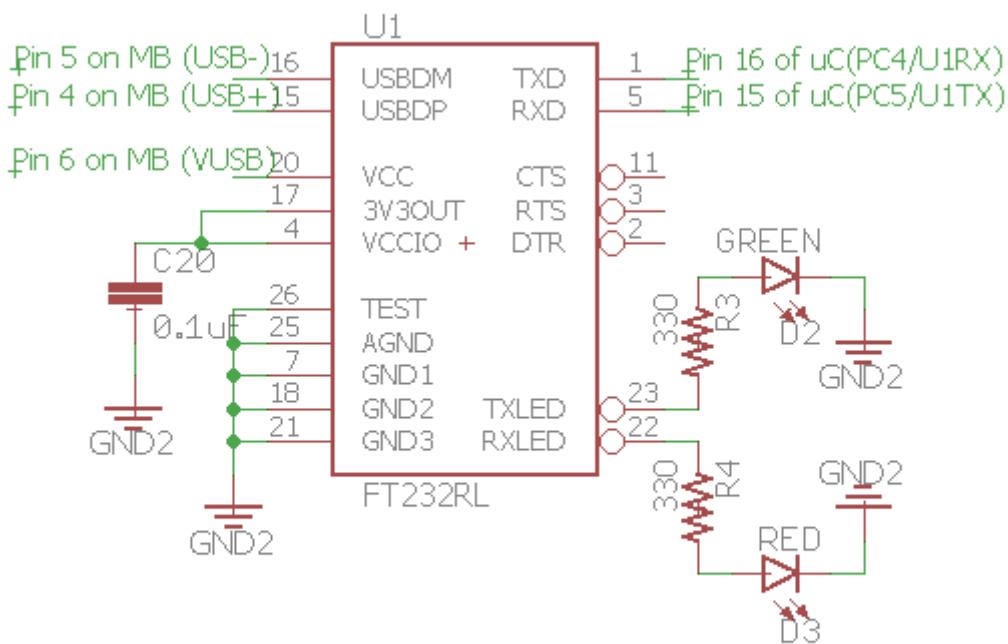
LCD can be interfaced in 8bit or 4 bit interfacing mode. In 8 bit mode it requires 3 control line and 8 data lines. To reduce number of I/Os required, Fire Bird V robot uses 4 bit interfacing mode which requires 2 control lines and 4 data lines. In this mode upper and lower nibble of the data/command byte needs to be sent separately. RW(Read/Write) control line of lcd is grounded so it can only work in write mode. The EN line is used to tell the LCD that microcontroller has sent data to it or microcontroller is ready to receive data from LCD. This is indicated by a high-to-low transition on this line. To send data to the LCD, program should make sure that this line is low (0) and then set the other two control lines as required and put data on the data bus. When this is done, make EN high (1) and wait for the minimum amount of time as specified by the LCD datasheet, and end by bringing it to low (0) again. When RS is low (0), data is treated as a command or special instruction by the LCD (such as clear screen, position cursor, etc.). When RS is high (1), data being sent is treated as text data which should be displayed on the screen.

**LCD Connections of Plug and Play board****LCD Connections of uC based board**

LCD	Pin Name(uC)	Pin Name(Plug and Play)
RS	PF0	PD6
EN	PF2	PD7
DB4	PD4	PA2
DB5	PD5	PA3
DB6	PD6	PA4
DB7	PD7	PA5

5.7 USB Communication

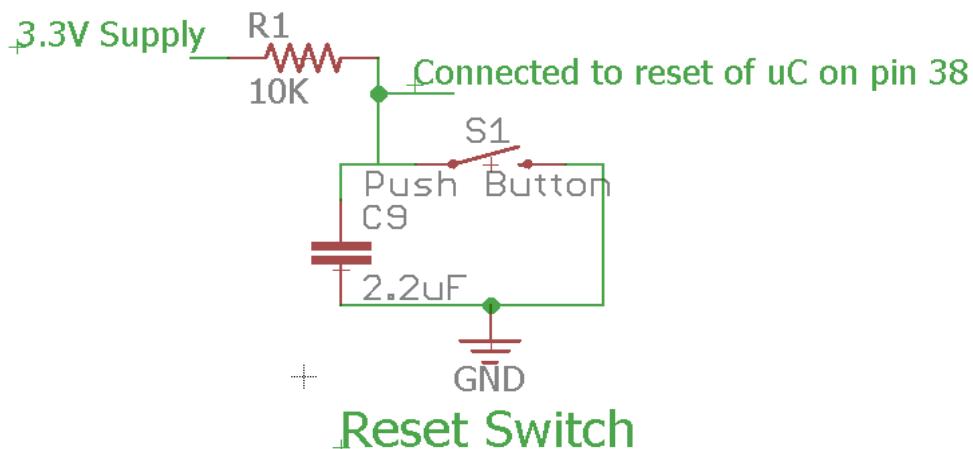
Fire Bird V's main board has USB port socket. Microcontroller accesses USB port via main board socket. All its pins are connected to the microcontroller adapter board via main board's socket connector. FT232 is a USB to TTL level serial converter. It is used for adding USB connectivity to the microcontroller adapter board. With onboard USB circuit Fire Bird V can communicate serially with the PC through USB port without the use of any external USB to Serial converter. Microcontroller socket uses USB port from the main board. Data transmission and reception is indicated using TX and RX LEDs which are located near the FT232 IC. This IC is only on the uC based board. Plug and play board has its own usb port on TIVA launcpad. The schematic of ft232 is shown below.



USB To Serial Convertor

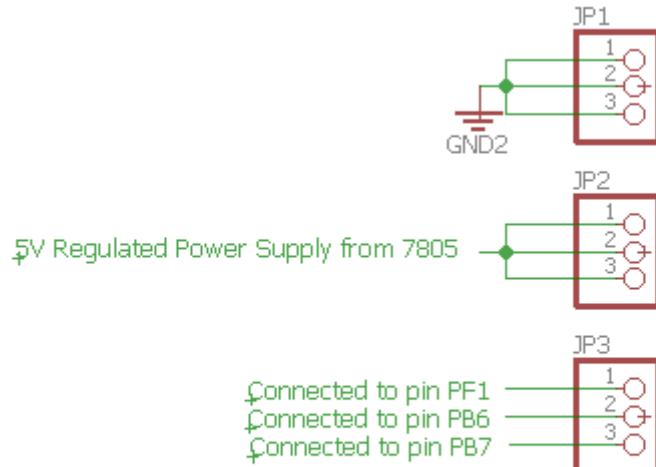
5.8 Reset Switch

The Plug and play board makes use of reset button present on the TIVA launchpad. The uC based has a switch connected to the reset the reset pin 38 of the microcontroller. The schematic is given below.

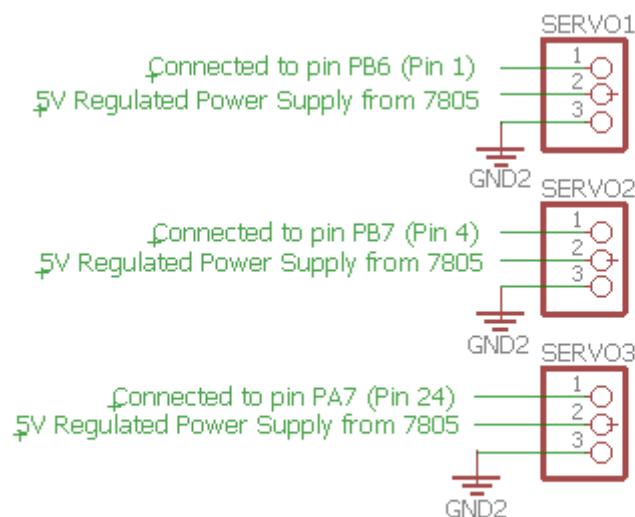


5.9 Servo Connectors

The microcontroller board has three Servo connectors. It can be used for driving servo motors of camera pod or any other attachment. Power for the servo connector is provided by the “5V servo supply” voltage regulator. Both the board have different pwm pins for servo which can be seen from the schematic.



Servo Connections for Plug and Play Board



Servo Connections for uC based Board

5.10 TM4C123GH6PM Micro-controller:

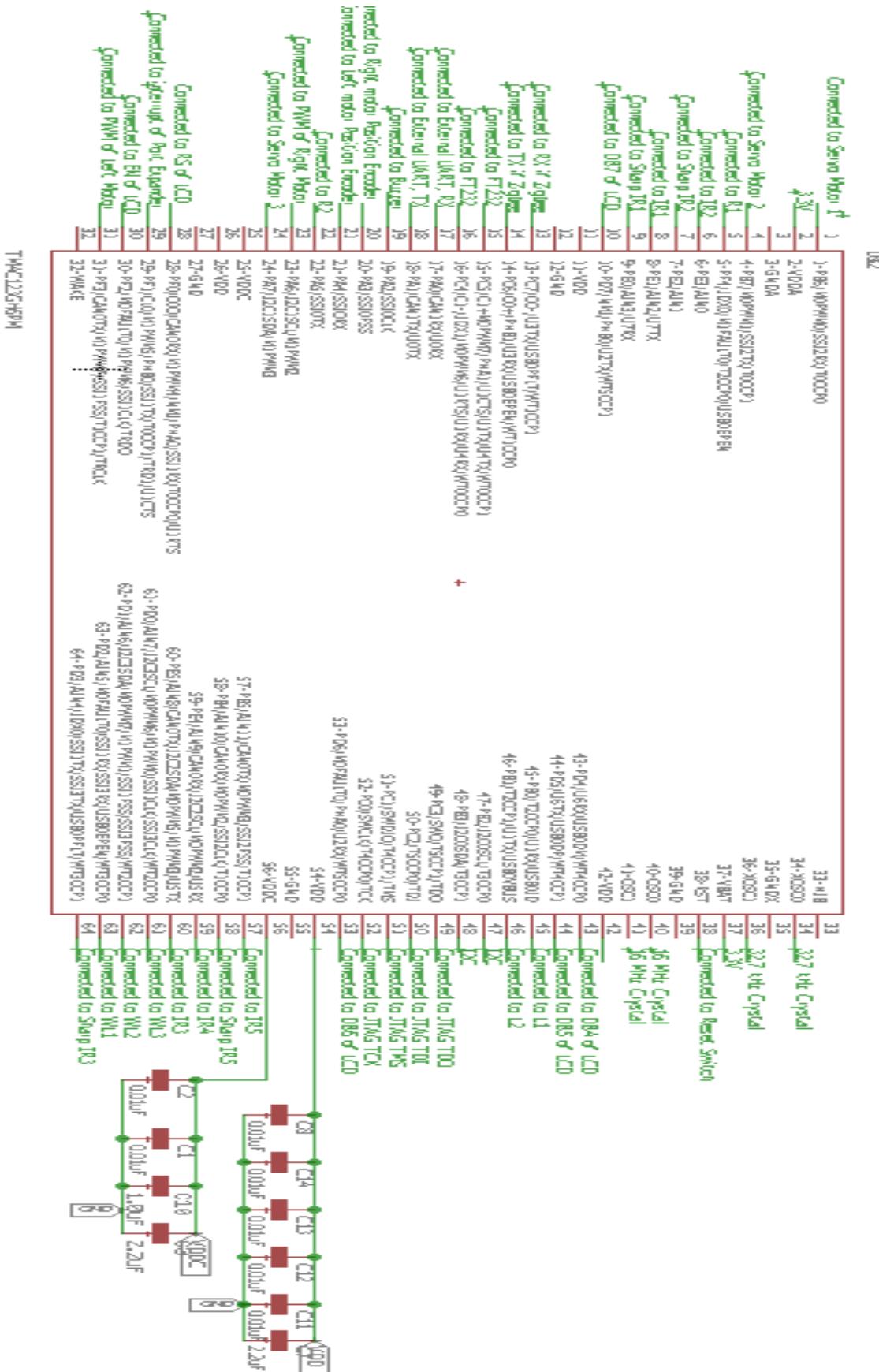
The Tiva™ C Series ARM Cortex-M4 microcontrollers provide top performance and advanced integration. The product family is positioned for cost-conscious applications requiring significant control processing and connectivity capabilities such as:

- Low power, hand-held smart devices
- Gaming equipment
- Home and commercial site monitoring and control
- Motion control
- Medical instrumentation
- Test and measurement equipment
- Factory automation
- Fire and security
- Smart Energy/Smart Grid solutions
- Intelligent lighting control
- Transportation

5.10.1 Feature of TM4C123GH6PM

- 32-bit ARM Cortex M4F architecture optimized for small-footprint embedded applications
- 80-MHz operation; 100 DMIPS performance
- 16-bit SIMD vector processing unit
- Harvard architecture characterized by separate buses for instruction and data
- Deterministic, high-performance interrupt handling for time-critical applications
- Enhanced system debug with extensive breakpoint and trace capabilities
- Ultra-low power consumption with integrated sleep modes

Schematic Of the connections is shown below.



5.11 Pin Functionality

5.11.1 Pin of uC

Pin No.	Pin	Complete Pin Connections
1	PB6	Servo Motor 1
2	VDDA	VDD filtered through capacitors
3	GNDA	Ground
4	PB7	Servo Motor 2
5	PF4	Pin 53 Right Motor 1
6	PE3	16 IR proximity Sensor 2
7	PE2	Output of Second OpAmp of lm324
8	PE1	12 IR Proximity Sensor 1
9	PE0	output of First OpAmp of lm324
10	PD7	28 DB7 of LCD Data
11	VDD	VDD filtered through capacitors
12	GND	Ground
13	PC7	13 Zigbee Rx
14	PC6	14 Zigbee Tx
15	PC5	FT232
16	PC4	FT232
17	PA0	External UART
18	PA1	External UART
19	PA2	Buzzer
20	PA3	63 Right Position Encoder Interrupt
21	PA4	64 Left Position Encoder Interrupt
22	PA5	55 Right Motor 2
23	PA6	54 Right Motor Pwm
24	PA7	Servo Motor 3
25	VDDC	Connected to VDDC on 56
26	VDD	VDD filtered through capacitors
27	GND	Ground
28	PF0	22 RS of LCD
29	PF1	INT A and B of to GPIO expander shorted and connected
30	PF2	24 EN of LCD
31	PF3	50 Left Motor PWM
32	WAKE	Ground
33	HIB	NC
34	XOSC0	32.7 KHz crystals(One End)
35	GNDX	Cap to crystal
36	XOSC1	32.7 KHz crystals(Other End)
37	VBAT	3.3 Volts
38	RST	Reset Switch
39	GND	Ground
40	OSC1	16 MHz crystal(One end)
41	OSC1	16 MHz crystal(Other end)
42	VDD	VDD filtered through capacitors
43	PD4	26 DB4 Of LCD
44	PD5	25 DB5 of LCD
45	PB0	51 Left Motor 1

46	PB1	52 Left Motor 2
47	PB2	I2C ADC SCL
48	PB3	I2C ADC SDA
49	PC3	JTAG TDO
50	PC2	JTAG TDI
51	PC1	JTAG TMS
52	PC0	JTAG TCK
53	PD6	27 DB6 Of LCD
54	VDD	VDD filtered through capacitors
55	GND	Ground
56	VDDC	Connected to VDDC on 25
57	PB5	46 IR Proximity Sensor 5
58	PB4	Output of First OpAmp of lm358
59	PE4	43 IR Proximity Sensor 4
60	PE5	42 IR Proximity Sensor 3
61	PD0	32 White Line Sensor 3
62	PD1	31 White Line Sensor 2
63	PD2	30 White Line Sensor 1
64	PD3	Output of Third OpAmp of lm324

5.11.2 Robot Main Board Connections

Pin Out	Pin Name	Functionality	PIN on uC DB	Pin on Pluggable DB
1	Current sensor	Current sense analog value	Not Using	Not Using
2	IR Proximity sensor 8	IR Proximity sensor 8 analog value	External Adc INT0	External Adc INT0
3	GND	Ground	Ground	Ground
4	DATA+	USB connection going to the AT-MEGA2560 USB connection with uC	PC4	
5	DATA-	microcontroller via FT232 USB to serial USB connection with uC	PC5	
6	VCC	USB converter. Connect TO VCC of FT232		
7	5V System	"5V System Voltage. Can be used for powering up any digital device with current limit of 400mA."		
8	5V System	"5V System Voltage. Can be used for powering up any digital device with current limit of 300mA."		
9	5V System	"5V System Voltage. Can be used for powering up any digital device with current limit of 300mA."		
10	5V System	"5V System Voltage. Can be used for powering up any digital device with current limit of 400mA."		
11	SHARP IR Range Sensor 1	Analog output of Sharp IR range Sensor 1	PE0(lm324 1)	PB4
12	IR Proximity Sensor 1	Analog output of IR Proximity sensor 1	PE1	PB5

13	XBee RXD	XBee wireless module Serial data in	PC7	PB1
14	XBee TXD	XBee wireless module Serial data out	PC6	PB0
15	SHARP IR Range Sensor 2	Analog output of Sharp IR range sensor 2	PE2(lm324 2)	External ADC INT1
16	IR Proximity Sensor 2	Analog output of IR Proximity sensor 2	PE3	PD0
17	RSSI	To capture the RSSI signal		
18	MOSI	MOSI of the Controller/NC create extra expansion headers		
19	MISO	MISO of controller/NC create extra expansion headers		
20	SCK	SCK of the controller/NC create extra expansion headers		
21	SSI	SS of the controller/ NC create extra expansion headers		
22	RS	connected to RS of LCD normal I/O	PF0	PD6
23	RW	connected to RW of LCD normal I/O	GND	GND
24	EN	connected to EN of LCD normal I/O	PF2	PD7
25	DB5	data pin of lcd normal I/O	PD5	PA3
26	DB4	data pin of lcd normal I/O	PD4	PA2
27	DB6	data pin of lcd normal I/O	PD6	PA1
28	DB7	data pin of lcd normal I/O	PD7	PA0
29	V Battery System	ADC to check the level of battery voltage		
30	WL1	Analog output of white line sensor 1	PD2	PE1
31	WL2	Analog output of white line sensor 2	PD1	PE2
32	WL3	Analog output of white line sensor 3	PD0	PE3
33	"Sharp IR Sensors 1and 5 Disable"			
34	IR Proximity Sensor Disable			
35	5V System	"5V system Voltage. Can be used for powering	up any digital device. Current Limit: 400mA."	
36	WL4	Analog output of white line sensor 4	External Adc INT1	External Adc INT2
37	WL5	Analog output of white line sensor 5	External Adc INT2	External Adc INT3
38	WL6	Analog output of white line sensor 6	External Adc INT3	External Adc INT4
39	WL7	Analog output of white line sensor 7	External Adc INT4	External Adc INT5
40	White Line Sensors Disable			
41	Sharp IR Range Finder 3	Analog output of Sharp IR range sensor 3	PD3 (lm 324 3)	PD2
42	IR Proximity Sensor 3	Analog output of IR Proximity sensor 3	PE5	PD3
43	IR Proximity Sensor 4	Analog output of IR Proximity sensor 4	PE4	PD1

44	Sharp IR Range Finder 4	Analog output of Sharp IR range sensor 4	in 5 ex (lm 324 5)	External Adc INT6
45	Sharp IR Range Finder 5	Analog output of Sharp IR range sensor 5	PB4 (lm358 1)	PE4
46	IR Proximity Sensor 5	Analog output of IR Proximity sensor 5	PB5	PE5
47	C11	motor not present		
48	C1	PWM not present		
49	c12	not present		
50	PWM L	left motor PWM(timer pin in PWM mode)	PF3	PF2
51	L1	left motor pin1 normal I/O	PB0	PF3
52	L2	left motor pin2 normal I/O	PB1	PB3
53	R1	right motor pin1 normal I/O	PF4	PC4
54	PWM R2	right motor PWM(timer pin in PWM mode)	PA6	PC5
55	R2	right motor pin2	PA5	PC6
56	NC			
57	NC			
58	NC			
59	NC			
60	NC			
61	NC			
62	Position encoder left	Output of Left position encoder (0-5V) PA4	PB2	
63	Position encoder right	Output of Right position encoder (0-5V) PA3	PF0	
64	position enocder C2	Output of C2 position encoder (0-5V)		
65	Position encoder C1	Output of C1 position encoder (0-5V)		
66	C22	NC		
67	C21	NC		
68	C2	Pwm	NC	
69	IR Prox6	Analog output of IR Proximity sensor 6 External Adc	INT6	External Adc INT7
70	IR Prox7	Analog output of IR Proximity sensor 7 External Adc	INT7	PE0
71	Buzzer	Input, V _{dd} 0.65V turns on the Buzzer	PA2	PF4
72	DAC Out	NC		
73	RS232 TX	NC		
74	RS232 RX	NC		

5.11.3 Pin Connection Of Plug And Play Board

Pin Name	Pin Connection on Main Board	Function
PA0		Used for Programming
PA1		Used for Programming
PA2	26	DB4 of LCD
PA3	27	DB5 of LCD
PA4	28	DB6 of LCD
PA5	29	DB7 of LCD
PA6		I2C

PA7		I2C
PB0	14	Zigbee Tx
PB1	13	Zigbee RX
PB2	62	Position encoder of left motor
PB3	52	L2
PB4	11	Sharp IR1
PB5	12	IR 1
PB6		Servo
PC0		
PC1		
PC2		
PC3		
PC4	53	R1
PC5	54	PWM of right motor
PC6	55	R2
PC7		Interrupt of port expander
PD0	16	IR 2
PD1	43	IR Prox 4
PD2	41	Sharp IR 3
PD3	42	IR Prox 3
PD4		
PD5		
PD6	22	RS of LCD
PD7	24	EN of LCD
PE0	70	IR 7
PE1	30	WL1
PE2	31	WL2
PE3	32	WL3
PE4	45	Sharp IR 5
PE5	46	IR 5
PE6		
PE7		
PF0	63	Position encoder of right motor
PF1		Servo
PF2	50	PWM of left motor
PF3	51	L1
PF4	71	Buzzer

6 Software Manual:

6.1 Code Composer Studio:

Code Composer Studio is an integrated development environment (IDE) that supports TI's Microcontroller and Embedded Processors portfolio. Code Composer Studio comprises a suite of tools used to develop and debug embedded applications. It includes an optimizing C/C++ compiler, source code editor, project build environment, debugger, profiler, and many other features. The intuitive IDE provides a single user interface taking you through each step of the application development flow. Familiar tools and interfaces allow users to get started faster than ever before. Code Composer Studio combines the advantages of the Eclipse software framework with advanced embedded debug capabilities from TI resulting in a compelling feature-rich development environment for embedded developers. This description is directly taken from the website of Texas Instruments and click to know more [about CC Studio](#)

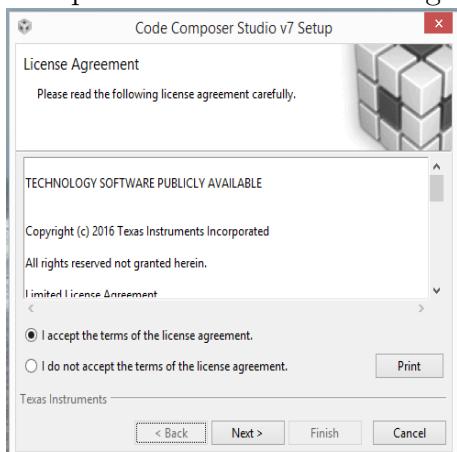
6.1.1 Download CC Studio:

At the time of writing this document Version 7 was the latest one. You can check for the latest at [Download CCS](#).(do not download any beta versions).There will be two installer files.The web installer will require Internet access until it completes. If the web installer version is unavailable or you can't get it to work, download, unzip and run the offline version. The offline download will be much larger than the installed size of CCS since it includes all the possible supported hardware.

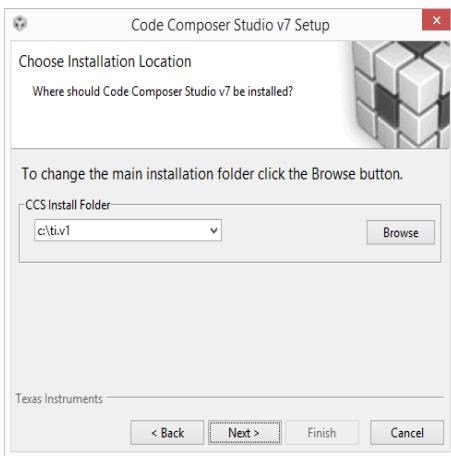
6.1.2 Installing C C Studio:

After the installer has started follow the steps mentioned below:

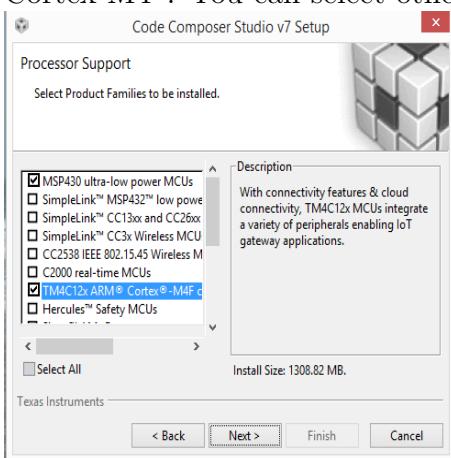
1. Accept the Software License Agreement and click Next.



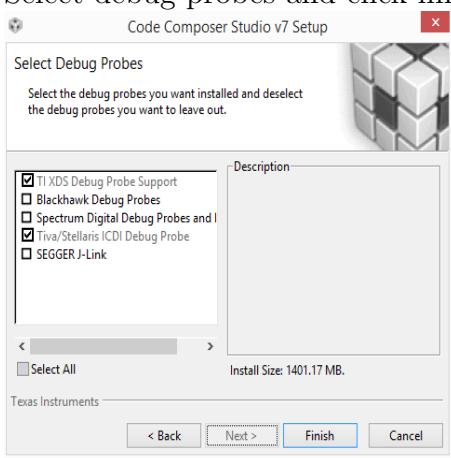
2. Select the destination folder and click next.



3. Select the processors that your CCS installation will support. You must select "TM4C12X Arm Cortex M4". You can select other architectures, but the installation time and size will increase.



4. Select debug probes and click finish



5. The installer process should take 15 - 30 minutes, depending on the speed of your connection. The offline installation should take 10 to 15 minutes. When the installation is complete, uncheck the "Launch Code Composer Studio v7" checkbox and then click Finish. There are several additional tools that require installation during the CCS install process. Click "Yes" or "OK" to proceed when these appear.
6. Install TivaWare for C Series (Complete). Download and install the latest full version of TivaWare from: [TivaWare](#). The filename is SW-TM4C-x.x.exe . This workshop was built using version 1.1. Your version may be a later one. If at all possible, please install TivaWare into the default location.

You can find additional information at these websites:

Main page: www.ti.com/launchpad

Tiva C Series TM4C123G LaunchPad:

<http://www.ti.com/tool/ek-tm4c123gxl>

TM4C123GH6PM folder:

<http://www.ti.com/product/tm4c123gh6pm>

BoosterPack webpage: www.ti.com/boosterpack

LaunchPad Wiki:

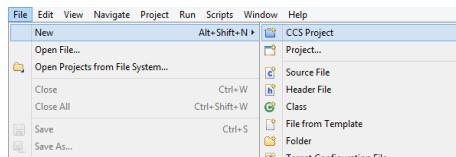
www.ti.com/launchpadwiki

For understanding the launchpad properly and to learn more about Tiva it is strongly recommended to go through the webpage **Tiva Workshops** and download and read the workbook

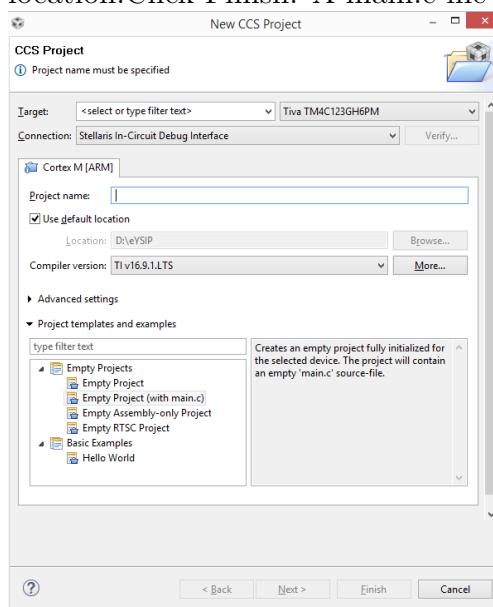
6.1.3 Create a New Project

To create new project follow the steps mentioned:

1. Click File then New and then CCS Projects



2. Select Target and connection as shown in the photo. Give a name to your project and save in a location. Click Finish. A main.c file will be open



6.1.4 Add Path and Build Variables

The path and build variables are used for:

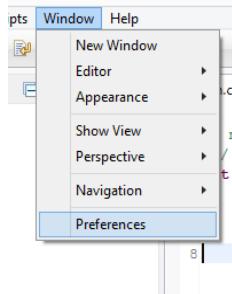
- Path variable – when you ADD (link) a file to your project, you can specify a "relative to" path. The default is PROJECT_LOC which means that your linked resource (like a .lib file) will be linked relative to your project directory.
- Build variable – used for items such as the search path for include files associated with a library – i.e. it is used when you build your project.

Variables can either have a PROJECT scope (that they only work for this project) or a WORKSPACE scope (that they work across all projects in the workspace). In the next step, we need to add (link) a library file and then add a search path for include files. First, we'll add these variables MANUALLY as WORKSPACE variables so that any project in your workspace can use the variables. Refer to the workbook by TI for adding as PROJECT

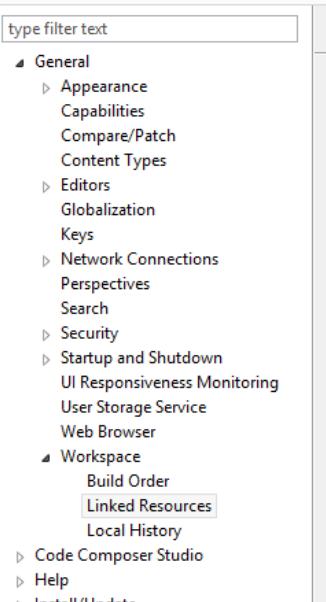
6.1.4.1 Adding a Path Variable

To add a path variable,:

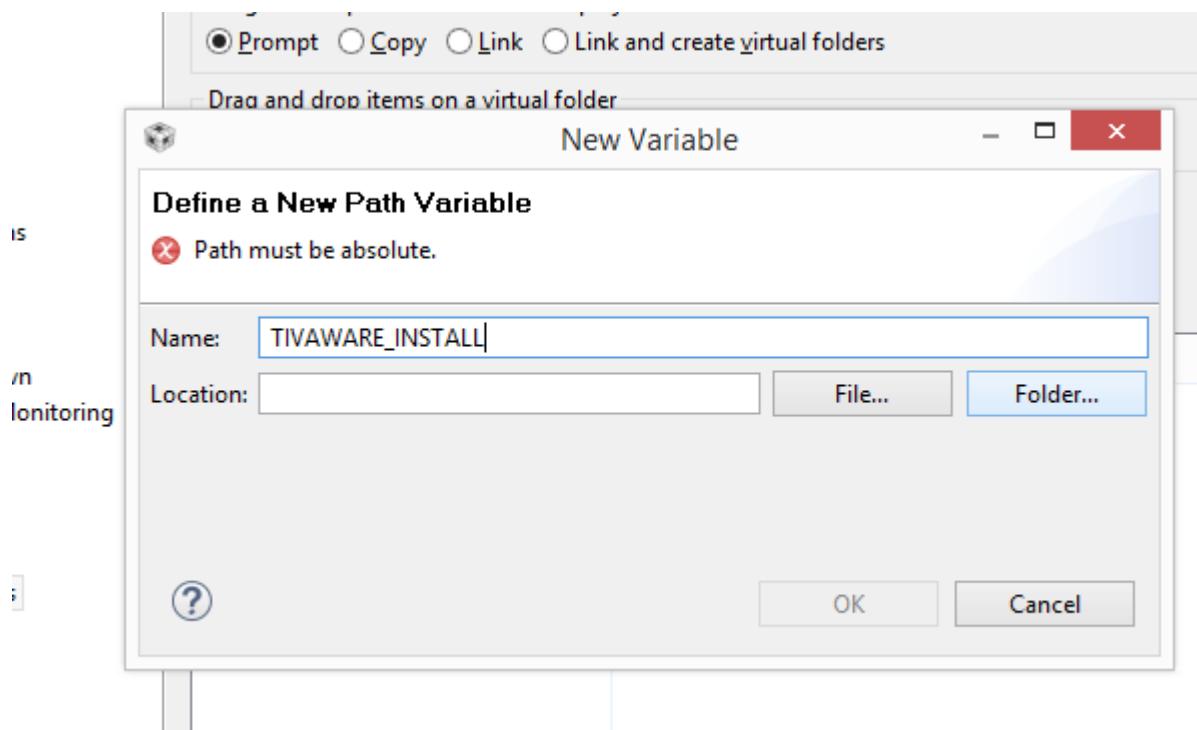
- Right-click on your Window Tab and select Preference.



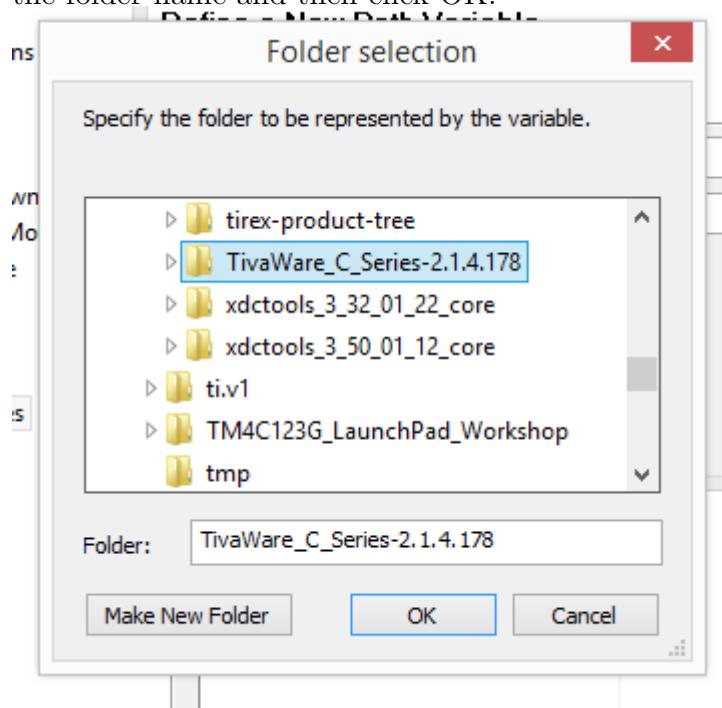
- Expand General list in the upper left-hand corner as shown and then expand the Resource list and click on Linked Resources: We want to add a New variable to specify exactly where you installed TivaWare.

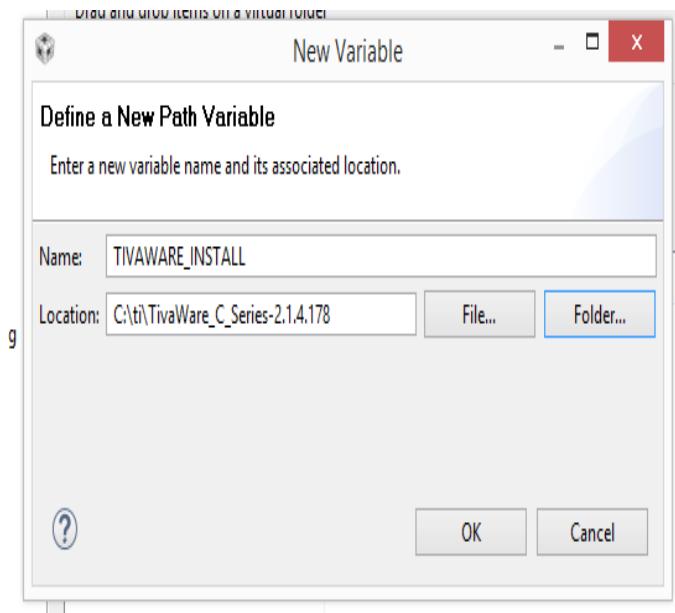


- Click New
- When the New Variable dialog appears, type TIVAWARE_INSTALL for the name.

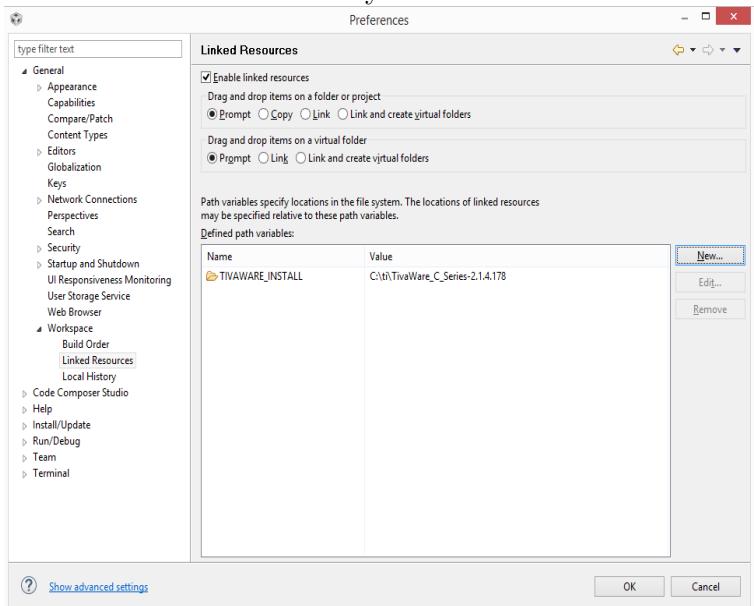


- For the Location, click the Folder... button and navigate to your TivaWare installation. Click on the folder name and then click OK.





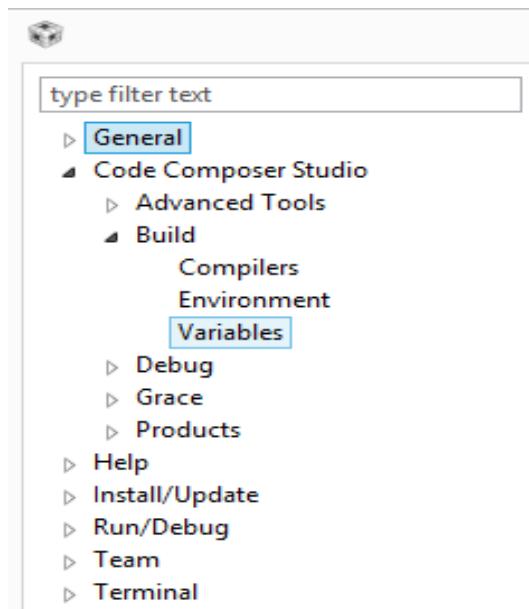
- Click OK. You should see your new variable listed in the Variables list.



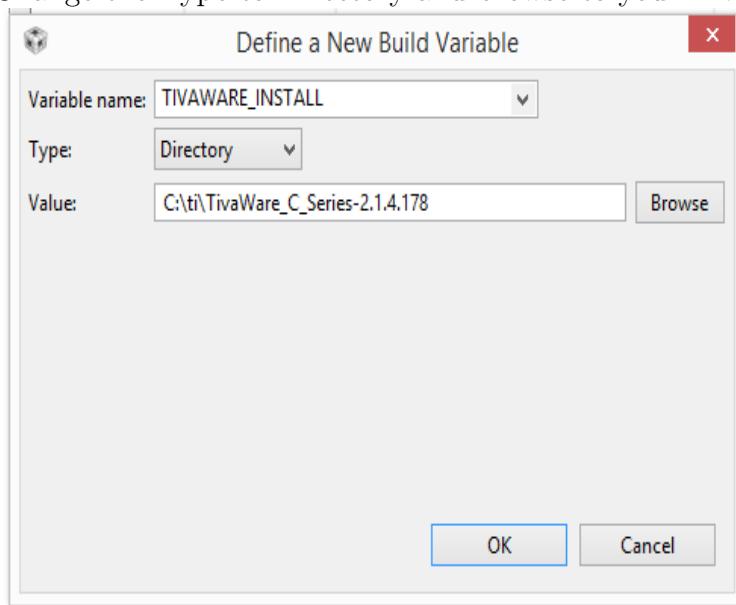
6.1.4.2 Adding a Build Variable

Now let's add a build variable that we will use in the include search path for the INCLUDE files associated with the TivaWare driver libraries.

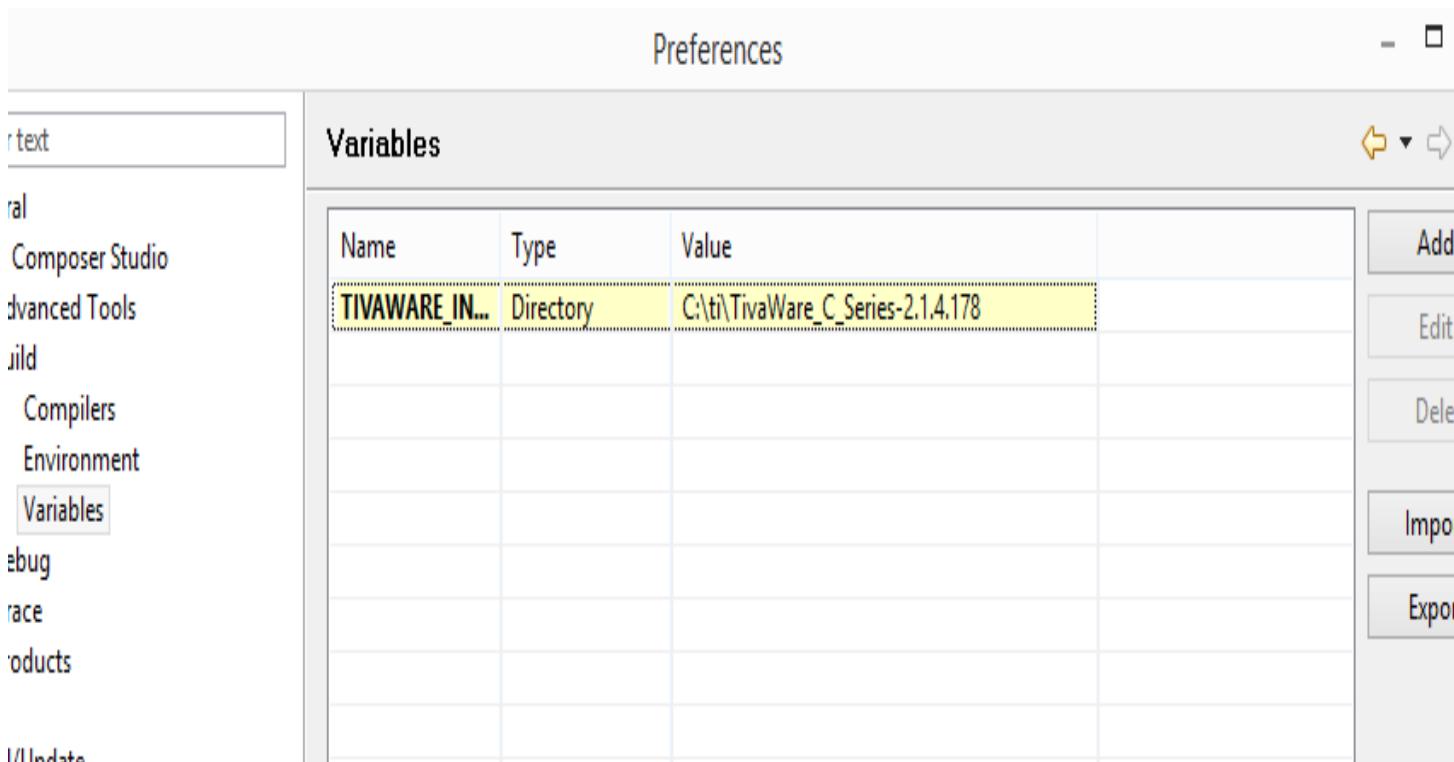
- Click on Code Composer Studio Build and then the Variables tab:



- Click the Add button. When the Define a New Build Variable dialog appears, insert TIVWARE_INSTALL into the Variables name box.
- Change the Type to Directory and browse to your Tivaware installation folder.



- Click OK.
- Click OK again to save and close the Build Properties window.

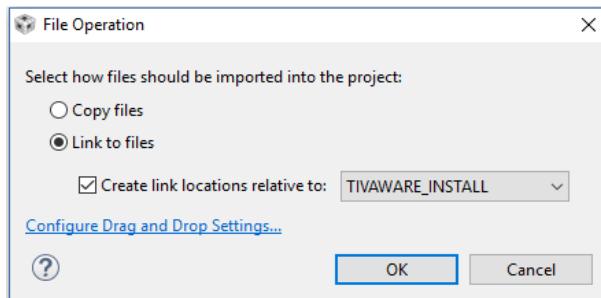


6.1.4.3 Adding driver.lib

We have to add the TivaWare driverlib.lib object library.

Right Click on Project, select Add Files and Navigate to:
 C:/TI/TivaWare_C_Series-1.1/driverlib/ccs/Debug/driverlib.lib

and click Open. The File Operation dialog will open



Use the TIVWARE_INSTALL path variable you created earlier. This means that the LINK (or reference to the library) file will be RELATIVE to the location of the TivaWare installation. If you hand this project to someone else, they can install the project anywhere in the file system and this link will still work. If you choose PROJECT_LOC, you would get a path that is relative to the location of your project and it would require the project to be installed at the same “level” in the directory structure. Another advantage of this approach is that if you wanted to link to a new version, say TivaWare_C_Series-1.2, all you have to do is modify the variable to the new folder name.

6.1.4.4 Build, Load, and Run

Assure that Daughter Board is connected to the laptop. Build and load your project to the TM4C123GH6PM flash memory by clicking the Debug button. If you ever want to build the project without loading it, click the HAMMER (Build) button.

Click the Resume button or press the F8 key on your keyboard.

6.2 Buzzer

Located in the folder “Buzzer_Beep” folder in the documentation. In this example, we will load buzzer beep code in Tiva based Fire Bird V. Now we will see in detail the structure of this code. This experiment demonstrates the simple operation of Buzzer ON/OFF with one second delay. Buzzer is connected to PORTF 4 pin of the Tiva Launchpad. If you have uC based board then it is connected to PORTA 2.

6.2.1 Code

```

1 #include <stdint.h>
2 #include <stdbool.h>
3 #include "inc/hw_types.h"
4 #include "inc/hw_memmap.h"
5 //This header File is important to Unlock GPIO Pins
6 #include "inc/hw_gpio.h"
7 #include "driverlib/sysctl.h"
8 #include "driverlib/gpio.h"

9
10 **** Useful Macros Definition*****
11 /***Remove the comments if you are using uC board*****
12 #define buzzerEnable    SYSCTL_PERIPH_GPIOA
13 #define buzzer          GPIO_PORTA_BASE
14 #define buzzerPin       GPIO_PIN_2
15 ****
16
17 /***Remove the comments if you are using uC board*****
18 #define buzzerEnable    SYSCTL_PERIPH_GPIOF
19 #define buzzer          GPIO_PORTF_BASE
20 #define buzzerPin       GPIO_PIN_4
21 ****
22
23 #define buzzerOn()      GPIOPinWrite(buzzer ,buzzerPin ,255)
24 #define buzzerOff()     GPIOPinWrite(buzzer ,buzzerPin ,0)
25 ****
26 void setupCLK();
27 void peripheralEnable();
28 void configIOPin();
29 void delay_ms(uint64_t delay);
30 void delay_ms(uint64_t delay);

31 int main(void) {
32     setupCLK();
33     peripheralEnable();
34     configIOPin();
35     while(1){
36         buzzerOn();
37         delay_ms(1000);
38         buzzerOff();
39         delay_ms(1000);
40     }
41 }
42 ****
43 * This function is used to setup Clock frequency of the controller
44 * It can be changed through codes
45 * In this we have set frequency as 40Mhz
46 * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
47 ****
48 void setupCLK(){
49     SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
50 }
51 ****
52 * Enabling System Peripherals
53 * buzzer Port in this case

```

```

55 * buzzerPin for buzzer output
56 *****/
57 void peripheralEnable(){
58     SysCtlPeripheralEnable(buzzerEnable);
59     /* Just in case you are not familiar with macros*****/
60     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
61     /* This is enabling PORTF*****/
62 }
63 *****
64 * Configuring Pin as Input Or Output
65 *****/
66 void configIOPin(){
67     GPIOPinTypeGPIOOutput(buzzer, buzzerPin);
68     /* Just in case you are not familiar with macros*****/
69     GPIOPinTypeGPIOOutput(buzzer, GPIO_PIN_4);
70     /* This is P4 output*****/
71 }
72 *****
73 * Calculating Delays
74 * extern void SysCtlDelay(uint32_t ui32Count)
75 * waits until the counting has been completed
76 *****/
77 void delay_ms(uint64_t delay){
78     SysCtlDelay(delay*SysCtlClockGet() /3000.0);
79 }
80 void delay_us(uint64_t delay){
81     SysCtlDelay(delay*SysCtlClockGet() /3000000.0);
82 }
83

```

6.3 Simple I/O Operation

This experiment demonstrates the simple I/O operations. This example is only for plug and play board, but this should not discourage the user from understanding the example as it provide very important example of I/O operation on TIVA platform.

In this lab you have to use switch SW1, SW2 and RGB LED present on Tiva C series board.

1. Use switch SW1 to Turn on Red LED on first switch press, Green LED on second switch press and Blue LED on third switch press. Repeat the same cycle next switch press onwards. Note that LED should remain on for the duration switch is kept pressed i.e. LED should turn off when switch is released.
2. Use switch SW2 and sw2Status (a variable). Your program should increment sw2Status by one, every time switch is pressed. Note how the value of sw2Status changes on each switch press. Use debugger and add sw2Status to "Watch Expression" window. (You will find Continuous Refresh button on top of the Expression Window). You can use step debugging or breakpoints to check the variable value. Hint: To add variable to Expression Window, select and right click the variable name and select Add Watch Expression". To view Expression Window, click on View button from CCS menu bar and select Expressions.

6.3.1 Code for Plug and Play Board

```

1 #include <stdint.h>
2 #include <stdbool.h>
3 #include "inc/hw_types.h"
4 #include "inc/hw_memmap.h"
5 #include "inc/hw_gpio.h" //To unlock locked pins for GPIO
6 #include "driverlib/sysctl.h"
7 #include "driverlib/gpio.h"
8 #define userSwitch1 GPIO_PIN_0
9 #define redLed      GPIO_PIN_1
10 #define blueLed    GPIO_PIN_2
11 #define greenLed   GPIO_PIN_3
12 #define userSwitch2 GPIO_PIN_4
13 #define LOCK_F (*((volatile unsigned long *)0x40025520))
14 #define CR_F   (*((volatile unsigned long *)0x40025524))
15 void setupCLK();
16 void configIOPin();
17 void delay_ms(uint64_t delay);
18 void delay_us(uint64_t delay);
19
20 int main()

```

```

21     setupCLK();
22     SysCtlDelay(3);
23     configIOPin();
24     unsigned char pinData=1;
25     unsigned char state=2;
26     unsigned char countSwitch2=0;
27     unsigned char flagSW1=0;
28     unsigned char flagSW2=0;
29     while(1){
30         pinData=GPIOPinRead(GPIO_PORTF_BASE, userSwitch2 | userSwitch1 );
31         if((pinData&0x01)==0)
32             flagSW1=1;
33         else if((flagSW1==1)&&(pinData&0x01)==1){
34             countSwitch2+=1;
35             flagSW1=0;
36         }
37         if((pinData&0x10)==0){
38             GPIOPinWrite(GPIO_PORTF_BASE, redLed | blueLed | greenLed , state );
39             flagSW2=1;
40         }
41         else if(((flagSW2==1)&&(pinData&0x10)==0x10)){
42             flagSW2=0;
43             GPIOPinWrite(GPIO_PORTF_BASE, redLed | blueLed | greenLed ,0 );
44             state=state *2;
45             if(state >8)
46                 state=2;
47         }
48         delay_ms(5);
49     }
50 }
51 /**
52 * This function is used to setup Clock frequency of the controller
53 * Enabling System Peripherals
54 * PORTF in this case
55 */
56 void setupCLK(){
57     SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
58     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
59 }
60 /**
61 * Configuring Pin as Input Or Output
62 * PF0 by default is locked and cannot
63 * be used as input unless it is unlocked
64 */
65 void configIOPin(){
66     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, redLed | blueLed | greenLed );
67     HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
68     HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
69     HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;
70     GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, userSwitch2 | userSwitch1 );
71     GPIOPadConfigSet(GPIO_PORTF_BASE ,userSwitch2 | userSwitch1 ,GPIO_STRENGTH_12MA,
72     GPIO_PIN_TYPE_STD_WPU);
73 }
74 /**
75 * Calculating Delays
76 */
77 void delay_ms(uint64_t delay){
78     SysCtlDelay(delay*(SysCtlClockGet() /3000));
79 }
80 void delay_us(uint64_t delay){
81     SysCtlDelay(delay*(SysCtlClockGet() /3000000UL));
82 }

```

6.4 Robot Direction Control

Located in the folder “Experiments Motion Control Simple” folder. Robot’s motors are controlled by L293D motor controller. Using L293D, microcontroller can control direction and velocity of both of the motors. To change the direction appropriate logic levels (High/Low) are applied to IC L293D’s direction pins. Velocity control is done using pulse width modulation (PWM) applied to Enable pins of L293D IC.

The Motor connections are as shown below

Motor Pin	Pin Name(uC)	Pin Name(Plug and Play)
L1	PB0	PF3
L2	PB1	PB3
PWM L	PF3	PF2
R1	PF4	PC4
R1	PA5	PC6
PWM R	PA6	PC5

6.4.1 Code for Plug and Play Board:

```

1 #include <stdint.h>
2 #include <stdbool.h>
3 #include "inc/hw_types.h"
4 #include "inc/hw_memmap.h"
5 //This header File is important to Unlock GPIO Pins
6 #include "inc/hw-gpio.h"
7 #include "driverlib/sysctl.h"
8 #include "driverlib/gpio.h"
9 //Used for controlling Motor direction
10 #define right 0x41
11 #define left 0x18
12 #define softRight 0x10
13 #define softLeft 0x01
14 #define forward 0x11
15 #define backward 0x48
16
17 void setupCLK();
18 void peripheralEnable();
19 void configIOPin();
20 void delay_ms(uint64_t delay);
21 void delay_us(uint64_t delay);
22 void motion(uint8_t);
23
24 int main(void) {
25     setupCLK();
26     peripheralEnable();
27     configIOPin();
28     while(1){
29         motion(forward);
30         delay_ms(1000);
31         motion(right);
32         delay_ms(1000);
33         motion(left);
34         delay_ms(1000);
35         motion(backward);
36         delay_ms(1000);
37     }
38 }
39 ****
40 * This function is used to setup Clock frequency of the controller
41 * It can be changed through codes
42 * In this we have set frequency as 40Mhz
43 * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
44 ****
45 void setupCLK(){
46     SysCtlClockSet(SYSCLOCK_SYSDIV_5|SYSCLOCK_USE_PLL|SYSCLOCK_XTAL_16MHZ|SYSCLOCK_OSC_MAIN);
47 }
48 ****
49 * Enabling System Peripherals
50 * PORTF,PORTB and PORTC in this case
51 ****
52 void peripheralEnable(){
53     SysCtlPeripheralEnable(SYSCLOCK_PERIPH_GPIOB);
54     SysCtlPeripheralEnable(SYSCLOCK_PERIPH_GPIOF);
55     SysCtlPeripheralEnable(SYSCLOCK_PERIPH_GPIOC);
56 }
57 ****
58 * Configuring Pin as Input Or Output
59 * And Setting PWM Pin to Always High
60 ****
61 void configIOPin(){
62     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_3);
63     GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6);
64     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_3|GPIO_PIN_2);
65     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 255);
66     GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 255);
67 }
```

```

68  ****
69 * Calculating Delays
70 ****
71 void delay_ms(uint64_t delay){
72     SysCtlDelay(delay*(SysCtlClockGet() /3000));
73 }
74 void delay_us(uint64_t delay){
75     SysCtlDelay(delay*(SysCtlClockGet() /3000000UL));
76 }
77 ****
78 * This function is for giving the direction of motion
79 * Macros have been defined at starting
80 * Macros for directions are 8 bits
81 * Out of these 8 bits only 4 are used
82 * Bit 0 (LSB) corresponds to PB3
83 * Bit 3 corresponds to PF3
84 * Bit 4 corresponds to PC4
85 * Bit 6 corresponds to PF6
86 ****
87 void motion(uint8_t direction){
88     GPIOPinWrite(GPIO_PORTB_BASE,GPIO_PIN_3,direction<<3);
89     GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_4|GPIO_PIN_6,direction);
90     GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3,direction);
91 }
92

```

6.4.2 Code for uC based Board:

```

1 #include <stdint.h>
2 #include <stdbool.h>
3 #include "inc/hw_types.h"
4 #include "inc/hw_memmap.h"
5 //This header File is important to Unlock GPIO Pins
6 #include "inc/hw-gpio.h"
7 #include "driverlib/sysctl.h"
8 #include "driverlib/gpio.h"
9 //Used for control Motor direction
10 #define right 0x22
11 #define left 0x11
12 #define softRight 0x02
13 #define softLeft 0x10
14 #define forward 0x12
15 #define backward 0x21
16 #define stop 0x00
17
18 void setupCLK();
19 void peripheralEnable();
20 void configIOPin();
21 void delay_ms(uint64_t delay);
22 void delay_ms(uint64_t delay);
23 void motion(uint8_t);
24
25 int main(void) {
26     setupCLK();
27     peripheralEnable();
28     configIOPin();
29     while(1){
30         motion(forward);
31         delay_ms(1000);
32         motion(right);
33         delay_ms(1000);
34         motion(left);
35         delay_ms(1000);
36         motion(backward);
37         delay_ms(1000);
38     }
39 }
40 ****
41 * This function is used to setup Clock frequency of the controller
42 * It can be changed through codes
43 * In this we have set frequency as 40Mhz
44 * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
45 ****
46 void setupCLK(){
47     SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
48 }

```

```

49  ****
50  * Enabling System Peripherals
51  * PORTF,PORTB and PORTC in this case
52  ****
53  void peripheralEnable(){
54      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
55      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
56      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
57  }
58  ****
59  * Configuring Pin as Input Or Output
60  * And Setting PWM Pin to Always High
61  ****
62  void configIOPin(){
63      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,GPIO_PIN_0|GPIO_PIN_1);
64      GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE,GPIO_PIN_6|GPIO_PIN_5);
65      GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_4|GPIO_PIN_3);
66      GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3,255);
67      GPIOPinWrite(GPIO_PORTA_BASE,GPIO_PIN_6,255);
68  }
69  ****
70  * Calculating Delays
71  ****
72  void delay_ms(uint64_t delay){
73     SysCtlDelay(delay*(SysCtlClockGet()/3000));
74 }
75  void delay_us(uint64_t delay){
76     SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
77 }
78  ****
79  * This function is for giving the direction of motion
80  * Macros have been defined at starting
81  * Macros for directions are 8 bits
82  * Out of these 8 bits only 4 are used
83  * Bit 0 (LSB) corresponds to PB0
84  * Bit 1 corresponds to PF1
85  * Bit 4 corresponds to PF4
86  * Bit 5 corresponds to PA5
87  ****
88  void motion(uint8_t direction){
89      GPIOPinWrite(GPIO_PORTB_BASE,GPIO_PIN_0|GPIO_PIN_1,direction);
90      GPIOPinWrite(GPIO_PORTA_BASE,GPIO_PIN_5,direction);
91      GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_4,direction);
92 }

```

6.5 Robot Position Control Using Interrupts

Position encoders give position / velocity feedback to the robot. It is used in closed loop to control robot's position and velocity. Position encoder consists of optical encoder and slotted disc assembly. When this slotted disc moves in between the optical encoder we get square wave signal whose pulse count indicates position and time period indicates velocity.

Connections:

Plug and Play Board:

PB2 : External interrupt for left motor position encoder

PF0 : External interrupt for the right position encoder

uC Based Board:

PA4 : External interrupt for left motor position encoder

PA3 : External interrupt for the right position encoder

6.5.1 Code for Plug and Play Board:

```

1 #include <stdint.h>
2 #include <stdbool.h>
3 #include "inc/tm4c123gh6pm.h"

```

```

4      #include "inc/hw_memmap.h"
5      #include "inc/hw_types.h"
6      #include "inc/hw_gpio.h"
7      #include "driverlib/sysctl.h"
8      #include "driverlib/interrupt.h"
9      #include "driverlib/gpio.h"
10     //This header File is important to Unlock GPIO Pins
11     #include "inc/hw_gpio.h"
12     #include "driverlib/sysctl.h"
13     #include "driverlib/gpio.h"
14
15     #define right          0x41
16     #define left           0x18
17     #define softRight      0x10
18     #define softLeft       0x01
19     #define forward        0x11
20     #define backward       0x48
21     #define stop           0x00
22
23     void setupCLK();
24     void peripheralEnable();
25     void configIOPin();
26     void delay_ms(uint64_t delay);
27     void delay_us(uint64_t delay);
28     void motion(uint8_t);
29     void interruptEnable();
30     void encoderInterruptEncountered();
31     void encoderInterruptEncountered1();
32     void angleRotate(uint16_t Degrees);
33     void linearDistanceMM(unsigned int DistanceInMM);
34     void rightDegrees(unsigned int Degrees);
35     void leftDegrees(unsigned int Degrees);
36     void forwardMM(unsigned int DistanceInMM);
37     void backwardMM(unsigned int DistanceInMM);
38     volatile unsigned long int ShaftCountRight = 0;
39     volatile unsigned long int ShaftCountLeft = 0;
40
41     int main(void) {
42         setupCLK();
43         peripheralEnable();
44         configIOPin();
45         interruptEnable();
46         while(1){
47             forwardMM(100);
48             delay_ms(1000);
49             leftDegrees(90);
50             delay_ms(1000);
51         }
52     }
53     ****
54     * This function is used to setup Clock frequency of the controller
55     * It can be changed through codes
56     * In this we have set frequency as 40Mhz
57     * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
58     ****
59     void setupCLK(){
60         SysCtlClockSet(SYSCLOCK_SYSDIV_5|SYSCLOCK_USE_PLL|SYSCLOCK_XTAL_16MHZ|SYSCLOCK_OSC_MAIN);
61     }
62     ****
63     * Enabling System Peripherals
64     * PORTF,PORTB and PORTC in this case
65     ****
66     void peripheralEnable(){
67         SysCtlPeripheralEnable(SYSCLOCK_PERIPH_GPIOB);
68         SysCtlPeripheralEnable(SYSCLOCK_PERIPH_GPIOF);
69         SysCtlPeripheralEnable(SYSCLOCK_PERIPH_GPIOC);
70     }
71     ****
72     * Configuring Pin as Input Or Output
73     * Unlocking PF0
74     * Setting PWM Pins to Always High
75     * Weak Pull to the Input Pins
76     ****
77     void configIOPin(){
78         GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,GPIO_PIN_3);
79         GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE,GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6);

```

```

80     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1);
81     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 255);
82     GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 255);
83     HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
84     HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
85     HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;
86     GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0);
87     GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_0, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
88     GPIOPinTypeGPIOInput(GPIO_PORTB_BASE, GPIO_PIN_2);
89     GPIOPadConfigSet(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
90 }
91 /**
92 * Calculating Delays
93 */
94 void delay_ms(uint64_t delay){
95     SysCtlDelay(delay*(SysCtlClockGet()/3000));
96 }
97 void delay_us(uint64_t delay){
98     SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
99 }
100 /**
101 * This function is for giving the direction of motion
102 * Macros have been defined at starting
103 * Macros for directions are 8 bits
104 * Out of these 8 bits only 4 are used
105 * Bit 0 (LSB) corresponds to PB3
106 * Bit 3 corresponds to PF3
107 * Bit 4 corresponds to PC4
108 * Bit 6 corresponds to PF6
109 */
110 void motion(uint8_t direction){
111     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, direction<<3);
112     GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_4|GPIO_PIN_6, direction);
113     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, direction);
114 }
115 /**
116 * For Enabling Interrupt on PORTF and PORTB*/
117 void interruptEnable(){
118     GPIOIntDisable(GPIO_PORTF_BASE, GPIO_PIN_0);
119     GPIOIntClear(GPIO_PORTF_BASE, GPIO_PIN_0);
120     GPIOIntRegister(GPIO_PORTF_BASE, encoderInterruptEncountered);
121     GPIOIntTypeSet(GPIO_PORTF_BASE, GPIO_PIN_0, GPIO_FALLING_EDGE);
122     GPIOIntEnable(GPIO_PORTF_BASE, GPIO_PIN_0);
123     GPIOIntDisable(GPIO_PORTB_BASE, GPIO_PIN_2);
124     GPIOIntClear(GPIO_PORTB_BASE, GPIO_PIN_2);
125     GPIOIntRegister(GPIO_PORTB_BASE, encoderInterruptEncountered1);
126     GPIOIntTypeSet(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_FALLING_EDGE);
127     GPIOIntEnable(GPIO_PORTB_BASE, GPIO_PIN_2);
128 }
129 /**
130 * Check on which pin of the PORTA has encountered an interrupt
131 * There is only one ISR for complete PORT
132 * No two PORTs can have same ISR
133 */
134 void encoderInterruptEncountered(){
135     if(GPIOIntStatus(GPIO_PORTF_BASE, false)&GPIO_PIN_0){
136         ShaftCountRight++;
137         GPIOIntClear(GPIO_PORTF_BASE, GPIO_PIN_0);
138     }
139     if(GPIOIntStatus(GPIO_PORTB_BASE, false)&GPIO_PIN_2){
140         ShaftCountLeft++;
141         GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 2);
142         GPIOIntClear(GPIO_PORTB_BASE, GPIO_PIN_2);
143     }
144 }
145 /**
146 * Function to Rotate to desired Angle
147 * Resolution can be Change to Get Higher Precision
148 */
149 void angleRotate(uint16_t Degrees){
150     unsigned long int ReqdShaftCountInt = 0;
151     ReqdShaftCountInt = Degrees/ 4.09;// division by resolution to get shaft count
152     ShaftCountRight = 0;
153     while (1){
154         if((ShaftCountRight>=ReqdShaftCountInt))
155             break;
156     }

```

```

156     motion(stop);
157 }
158 /**************************************************************************
159 * Function to Move in a Linear Distance
160 * Resolution can be Change to Get Higher Precision
161 **************************************************************************/
162 void linearDistanceMM( unsigned int DistanceInMM){
163     unsigned long int RreqdShaftCountInt = 0;
164     RreqdShaftCountInt =DistanceInMM / 5.338;;
165     ShaftCountRight=0;
166     ShaftCountLeft=0;
167     while(1){
168         if((ShaftCountRight >=RreqdShaftCountInt)&&(ShaftCountLeft >= RreqdShaftCountInt))
169             break;
170         else if((ShaftCountRight > RreqdShaftCountInt))
171             motion(softRight);
172         else if((ShaftCountLeft > RreqdShaftCountInt))
173             motion(softLeft);
174     }
175     motion(stop); //Stop robot
176 }
177 void forwardMM( unsigned int DistanceInMM){
178     motion(forward);
179     linearDistanceMM(DistanceInMM);
180 }
181
182 void backwardMM( unsigned int DistanceInMM){
183     motion(backward);
184     linearDistanceMM(DistanceInMM);
185 }
186 void leftDegrees( unsigned int Degrees){
187     motion(left); //Turn left
188     angleRotate(Degrees);
189 }
190 void rightDegrees( unsigned int Degrees){
191     motion(right); //Turn right
192     angleRotate(Degrees);
193 }
194

```

6.5.2 Code for uC based Board:

```

1 #include <stdint.h>
2 #include <stdbool.h>
3 #include "inc/tm4c123gh6pm.h"
4 #include "inc/hw_memmap.h"
5 #include "inc/hw_types.h"
6 #include "inc/hw_gpio.h"
7 #include "driverlib/sysctl.h"
8 #include "driverlib/interrupt.h"
9 #include "driverlib/gpio.h"
10
11 #define right          0x22
12 #define left           0x11
13 #define softRight      0x02
14 #define softLeft       0x10
15 #define forward        0x12
16 #define backward       0x21
17 #define stop           0x00
18
19 void setupCLK();
20 void peripheralEnable();
21 void gpioEnable();
22 void interruptEnable();
23 void encoderInterruptEncountered();
24 void linearDistanceMM( unsigned int);
25 void angleRotate( uint16_t );
26 void forwardMM( unsigned int );
27 void backwardMM( unsigned int );
28 void leftDegrees( unsigned int );
29 void rightDegrees( unsigned int );
30 void delay_ms( uint64_t delay );
31 void delay_us( uint64_t delay );
32 void motion( uint8_t direction );
33 volatile uint16_t ShaftCountRight=0,ShaftCountLeft=0;
34

```

```

35     int main(void) {
36         setupCLK();
37         peripheralEnable();
38         gpioEnable();
39         interruptEnable();
40         while(1){
41             forwardMM(100);
42             delay_ms(1000);
43             rightDegrees(90);
44             delay_ms(1000);
45         }
46     }
47
48 /***** This function is used to setup Clock frequency of the controller *****
49 * This function is used to setup Clock frequency of the controller
50 * It can be changed through codes
51 * In this we have set frequency as 40Mhz
52 * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
53 *****/
54 void setupCLK(){
55     SysCtlClockSet(SYSCLOCK_SYSCLK_5|SYSCLOCK_USE_PLL|SYSCLOCK_XTAL_16MHZ|SYSCLOCK_OSC_MAIN);
56 }
57
58 /***** Enabling System Peripherals *****
59 * Enabling System Peripherals
60 * PORTF,PORTB and PORTA in this case
61 *****/
62 void peripheralEnable(){
63     SysCtlPeripheralEnable(SYSCLOCK_PERIPH_GPIOB);
64     SysCtlPeripheralEnable(SYSCLOCK_PERIPH_GPIOF);
65     SysCtlPeripheralEnable(SYSCLOCK_PERIPH_GPIOA);
66 }
67
68 /***** Configuring Pin as Input Or Output *****
69 * Configuring Pin as Input Or Output
70 * Setting PWM Pins to Always High
71 * Weak Pull to the Input Pins
72 *****/
73 void gpioEnable(){
74     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_1|GPIO_PIN_2);
75     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,GPIO_PIN_0|GPIO_PIN_1);
76     GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE,GPIO_PIN_6|GPIO_PIN_5);
77     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_4|GPIO_PIN_3);
78     GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3,255);
79     GPIOPinWrite(GPIO_PORTA_BASE,GPIO_PIN_6,255);
80     GPIOPinTypeGPIOInput(GPIO_PORTA_BASE, GPIO_PIN_4|GPIO_PIN_3);
81     GPIOPadConfigSet(GPIO_PORTA_BASE ,GPIO_PIN_4|GPIO_PIN_5,GPIO_STRENGTH_2MA,GPIO_PIN_TYPE_STD_WPU
82 );
83 }
84 /****For Enabling Interrupt on PortA****/
85 void interruptEnable(){
86     GPIOIntDisable(GPIO_PORTA_BASE,GPIO_PIN_4|GPIO_PIN_3);
87     GPIOIntClear(GPIO_PORTA_BASE, GPIO_PIN_4|GPIO_PIN_3);
88     GPIOIntRegister(GPIO_PORTA_BASE, encoderInterruptEncountered);
89     GPIOIntTypeSet(GPIO_PORTA_BASE,GPIO_PIN_4|GPIO_PIN_3,GPIO_FALLING_EDGE);
90     GPIOIntEnable(GPIO_PORTA_BASE,GPIO_PIN_4|GPIO_PIN_3);
91 }
92 /**** ISR For External Interrupt on PortA*****/
93 * Check on which pin of the PORTA has encountered an interrupt
94 * There is only one ISR for complete PORT
95 * No two PORTs can have same ISR
96 *****/
97 void encoderInterruptEncountered(){
98     if(GPIOIntStatus(GPIO_PORTA_BASE, false)&GPIO_PIN_4){
99         ShaftCountLeft++;
100        GPIOIntClear(GPIO_PORTA_BASE, GPIO_PIN_4);
101    }
102    if(GPIOIntStatus(GPIO_PORTA_BASE, false)&GPIO_PIN_3){
103        ShaftCountRight++;
104        GPIOIntClear(GPIO_PORTA_BASE, GPIO_PIN_3);
105    }
106 }
107 /***** Calculating Delays *****/
108 void delay_ms(uint64_t delay){
109

```

```

110     SysCtlDelay( delay*(SysCtlClockGet() /3000));
111 }
112 void delay_us(uint64_t delay){
113     SysCtlDelay( delay*(SysCtlClockGet() /3000000UL));
114 }
115 /************************************************************************
116 * This function is for giving the direction of motion
117 * Macros have been defined at starting
118 * Macros for directions are 8 bits
119 * Out of these 8 bits only 4 are used
120 * Bit 0 (LSB) corresponds to PB3
121 * Bit 3 corresponds to PF3
122 * Bit 4 corresponds to PC4
123 * Bit 6 corresponds to PF6
124 *****/
125 void motion(uint8_t direction){
126     GPIOPinWrite(GPIO_PORTB_BASE,GPIO_PIN_0|GPIO_PIN_1,direction);
127     GPIOPinWrite(GPIO_PORTA_BASE,GPIO_PIN_5,direction);
128     GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_4,direction);
129 }
130 *****/
131 * Function to Rotate to desired Angle
132 * Resolution can be Change to Get Higher Precision
133 *****/
134 void angleRotate(uint16_t Degrees){
135     unsigned long int ReqdShaftCountInt = 0; // division by resolution to get shaft count
136     ReqdShaftCountInt = Degrees/ 4.09;;
137     ShaftCountRight = 0;
138     ShaftCountLeft = 0;
139     while (1){
140         if((ShaftCountRight>=ReqdShaftCountInt)&&(ShaftCountLeft>=ReqdShaftCountInt))
141             break;
142     }
143     motion(stop);
144 }
145 *****/
146 * Function to Move in a Linear Distance
147 * Resolution can be Change to Get Higher Precision
148 *****/
149 void linearDistanceMM(unsigned int DistanceInMM){
150     unsigned long int ReqdShaftCountInt = 0;
151     ReqdShaftCountInt =DistanceInMM / 5.338;;
152     ShaftCountRight=0;
153     ShaftCountLeft=0;
154     while(1){
155         if((ShaftCountRight > ReqdShaftCountInt)&&(ShaftCountLeft > ReqdShaftCountInt))
156             break;
157         else if((ShaftCountRight > ReqdShaftCountInt))
158             motion(softRight);
159         else if((ShaftCountLeft > ReqdShaftCountInt))
160             motion(softLeft);
161     }
162     motion(stop); //Stop robot
163 }
164 void forwardMM(unsigned int DistanceInMM){
165     motion(forward);
166     linearDistanceMM(DistanceInMM);
167 }
168 void backwardMM(unsigned int DistanceInMM){
169     motion(backward);
170     linearDistanceMM(DistanceInMM);
171 }
172 void leftDegrees(unsigned int Degrees){
173     motion(left); //Turn left
174     angleRotate(Degrees);
175 }
176 void rightDegrees(unsigned int Degrees){
177     motion(right); //Turn right
178     angleRotate(Degrees);
179 }
180

```

6.6 Timers and its Interrupts

The TM4C123GH6PM General-Purpose Timer Module (GPTM) contains six 16/32-bit GPTM blocks

and six 32/64-bit Wide GPTM blocks. Each 16/32-bit GPTM block provides two 16-bit timers/counters (referred to as Timer A and Timer B) that can be configured to operate independently as timers or event counters, or concatenated to operate as one 32-bit timer or one 32-bit Real-Time Clock (RTC). Each 32/64-bit Wide GPTM block provides 32-bit timers for Timer A and Timer B that can be concatenated to operate as a 64-bit timer.

Timers are mainly used for

- Velocity Control
- Servo Motor Control
- Event Scheduling
- Velocity Calculation

In this section the event scheduling application of timer is explained. To illustrate this the buzzer is switched On and OFF periodically. The remaining applications are explained in the further sections.

6.6.1 Code

```

1 #include <stdint.h>
2 #include <stdbool.h>
3 #include "inc/hw_types.h"
4 #include "inc/hw_memmap.h"
5 #include "inc/tm4c123gh6pm.h"
6 //This header File is important to Unlock GPIO Pins
7 #include "inc/hw_gpio.h"
8 #include "driverlib/sysctl.h"
9 #include "driverlib/gpio.h"
10 //Used for enabling the timer
11 #include "driverlib/timer.h"
12 //Used for enabling interrupt
13 #include "driverlib/interrupt.h"
14 //**** Useful Macros Definition*****
15 //*****Remove the comments if you are using uC board*****
16 #define buzzerEnable    SYSCTL_PERIPH_GPIOA
17 #define buzzer        GPIO_PORTA_BASE
18 #define buzzerPin      GPIO_PIN_2
19 ****
20 //*****Remove the comments if you are using uC board*****
21 #define buzzerEnable    SYSCTL_PERIPH_GPIOF
22 #define buzzer        GPIO_PORTF_BASE
23 #define buzzerPin      GPIO_PIN_4
24 ****
25 ****
26 #define buzzerOn()      GPIOPinWrite(buzzer ,buzzerPin ,255)
27 #define buzzerOff()     GPIOPinWrite(buzzer ,buzzerPin ,0)
28 ****
29 void setupCLK();
30 void peripheralEnable();
31 void configIOPin();
32 void timerEnable();
33
34 uint32_t ui32Period;    // used for generating one second delay
35 volatile int flag = 0;      //used to monitor the state of buzzer
36
37 int main(void) {
38     setupCLK();
39     peripheralEnable();
40     configIOPin();
41     timerEnable();
42     flag = 0;
43     while(1){
44         }
45     }
46 ****
47 * This function is used to setup Clock frequency of the controller

```

```

49 * It can be changed through codes
50 * In this we have set frequency as 40Mhz
51 * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
52 ****
53 void setupCLK(){
54     SysCtlClockSet(SYSCLOCK_SYSDIV_5|SYSCLOCK_USE_PLL|SYSCLOCK_XTAL_16MHZ|SYSCLOCK_OSC_MAIN);
55 }
56 ****
57 * Enabling System Peripherals
58 * buzzer Port in this case
59 * buzzerPin for buzzer output
60 * Enabling Timer 0
61 ****
62 void peripheralEnable(){
63     SysCtlPeripheralEnable(buzzerEnable);
64     SysCtlPeripheralEnable(SYSCLOCK_PERIPH_TIMER0);
65     //***** Just in case you are not familiar with macros*****
66     SysCtlPeripheralEnable(SYSCLOCK_PERIPH_GPIOF);
67     *****This is enabling PORTF*****
68 }
69 ****
70 * Configuring Pin as Input Or Output
71 ****
72 void configIOPin(){
73     GPIOPinTypeGPIOOutput(buzzer, buzzerPin);
74 }
75 ****
76 * Enabling Timer 0
77 * Timer is configured to be generate
78 interrupt every second
79 * Here sysCtlClockGet() is divided
80 by the on time of buzzer
81 ****
82 void timerEnable(){
83     TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
84     ui32Period = (SysCtlClockGet() / 1) / 2;
85     TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period - 1);
86     IntEnable(INT_TIMER0A);
87     TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
88     IntMasterEnable();
89     TimerEnable(TIMER0_BASE, TIMER_A);
90 }
91 ****
92 * This function is executed when the timer overflows
93 * In this example the buzzer is switched on and off alternatively
94 ****
95 void Timer0IntHandler(void)
96 {
97     // Clear the timer interrupt
98     TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
99     flag = !flag;
100    if(flag == 0){
101        buzzerOn();
102    }
103    else{
104        buzzerOff();
105    }
106 }
107

```

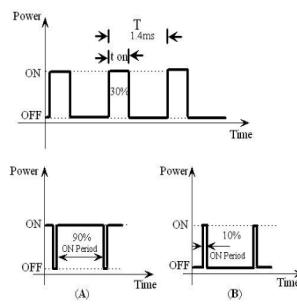
6.7 Robot Speed Control

6.7.1 Pulse Width Modulation(PWM)

Pulse width modulation is a process in which duty cycle of constant frequency square wave is modulated to control power delivered to the load i.e. motor.

Duty cycle is the ratio of ‘T-ON/ T’. Where ‘T-ON’ is ON time and ‘T’ is the time period of the wave. Power delivered to the motor is proportional to the ‘T-ON’ time of the signal. In case of PWM the motor reacts to the time average of the signal.

PWM is used to control total amount of power delivered to the load without power losses which generally occur in resistive methods of power control.



Above figure shows the PWM waveforms for motor velocity control. In case (A), ON time is 90 percent of time period. This wave has more average value. Hence more power is delivered to themotor. In case (B), the motor will run slower as the ON time is just 10 percent of time period.

The TM4C123GH6PM microcontroller contains two PWM modules, each with four PWM generator blocks and a control block, for a total of 16 PWM outputs. The control block determines the polarity of the PWM signals, and which signals are passed through to the pins. The connections of PWM motor pins are given in the section 6.7. The same code is modified to change the velocity of the motors.

6.7.2 Code for Plug and Play Board:

```

1  include <stdint.h>
2  #include <stdbool.h>
3  #include "inc/hw_types.h"
4  #include "inc/hw_memmap.h"
5  #include "driverlib/pin_map.h"
6  //This header File is important to Unlock GPIO Pins
7  #include "inc/hw_gpio.h"
8  #include "driverlib/sysctl.h"
9  #include "driverlib/gpio.h"
10 //Used for PWM
11 #include "driverlib/pwm.h"
12
13 #define right          0x41
14 #define left           0x18
15 #define softRight      0x10
16 #define softLeft       0x01
17 #define forward        0x11
18 #define backward       0x48
19 #define stop           0x00
20
21 void setupCLK();
22 void peripheralEnable();
23 void configIOPin();
24 void delay_ms(uint64_t delay);
25 void motion(uint8_t);
26 void enablePWM();
27 void Velocity(uint8_t lSpeed, uint8_t rSpeed);
28
29 int main(void) {
30     setupCLK();
31     peripheralEnable();
32     configIOPin();
33     enablePWM();
34     while(1){
35         Velocity(150, 150);
36         motion(forward);
37         delay_ms(2000);
38         motion(stop);
39         delay_ms(500);
40         Velocity(255, 255);
41         motion(backward);
42         delay_ms(800);
43         motion(stop);
44         delay_ms(500);
45         Velocity(255, 255);
46         motion(right);
47         delay_ms(1000);
48 }
```

```

49         motion(stop);
50         delay_ms(500);
51         Velocity(150, 150);
52         motion(left);
53         delay_ms(1000);
54         motion(stop);
55         delay_ms(500);
56         Velocity(150, 150);
57         motion(backward);
58         delay_ms(1000);
59     }
60 }
61 /**************************************************************************
62 * This function is used to setup Clock frequency of the controller
63 * It can be changed through codes
64 * In this we have set frequency as 40Mhz
65 * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
66 * The PWM module is clocked by the system clock through a divider, and that divider has
67 * a range of 2 to 64.
68 * By setting the divider to 64, it will run the PWM clock at 625 kHz.
69 **************************************************************************/
70 void setupCLK(){
71     SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
72     SysCtlPWMClockSet(SYSCTL_PWMDIV_64); //625kHz PWM Clock
73 }
74 /**************************************************************************
75 * Enabling System Peripherals
76 * PORTF,PORTB and PORTC in this case
77 **************************************************************************/
78 void peripheralEnable(){
79     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
80     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
81     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
82     SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0); // Enabling PWM0
83     SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1); // Enabling PWM1
84 }
85 /**************************************************************************
86 * Configuring Pin as Input Or Output
87 * And Setting PWM Pin to Always High
88 **************************************************************************/
89 void configIOPin(){
90     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_3);
91     GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6);
92     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_3|GPIO_PIN_2);
93 }
94 /**************************************************************************
95 * Calculating Delays
96 **************************************************************************/
97 void delay_ms(uint64_t delay){
98     SysCtlDelay(delay*(SysCtlClockGet()/3000));
99 }
100 void delay_us(uint64_t delay){
101     SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
102 }
103 /**************************************************************************
104 * This function is for giving the direction of motion
105 * Macros have been defined at starting
106 * Macros for directions are 8 bits
107 * Out of these 8 bits only 4 are used
108 * Bit 0 (LSB) corresponds to PB3
109 * Bit 3 corresponds to PF3
110 * Bit 4 corresponds to PC4
111 * Bit 6 corresponds to PF6
112 **************************************************************************/
113 void motion(uint8_t direction){
114     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, direction<<3);
115     GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_4|GPIO_PIN_6, direction);
116     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, direction);
117 }
118 /**************************************************************************
119 * This function is for enabling the PWM Modules
120 * PWM can be enabled on a pin based on the datasheet
121 **************************************************************************/
122 void enablePWM(){
123     GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_2);
124     GPIOPinConfigure(GPIO_PF2_M1PWM6);

```

```

125     GPIOPinTypePWM(GPIO_PORTC_BASE, GPIO_PIN_5);
126     GPIOPinConfigure(GPIO_PC5_M0PWM7);
127     //Count Down Mode
128     PWMGenConfigure(PWM0_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
129     PWMGenPeriodSet(PWM0_BASE, PWM_GEN_3, 255); //Load Count value
130     //Count Down Mode
131     PWMGenConfigure(PWM1_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
132     PWMGenPeriodSet(PWM1_BASE, PWM_GEN_3, 255); //Load Count value
133     PWMGenEnable(PWM0_BASE, PWM_GEN_3);
134     PWMGenEnable(PWM1_BASE, PWM_GEN_3);
135     PWMOutputState(PWM1_BASE, PWM_OUT_6_BIT, true);
136     PWMOutputState(PWM0_BASE, PWM_OUT_7_BIT, true);
137 }
138 /**************************************************************************
139 * This function is used to control the speed of the motors
140 * The speed can changed by the PWMPulseWidthSet() function
141 * lSpeed is used to control the speed of left motor
142 * rSpeed is used to control the speed of right motor
143 **************************************************************************/
144 void Velocity(uint8_t lSpeed, uint8_t rSpeed){
145     lSpeed=(lSpeed>255)?255:lSpeed;
146     rSpeed=(rSpeed>255)?255:rSpeed;
147     PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6, lSpeed);
148     PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7, rSpeed);
149 }
150

```

6.7.3 Code for uC based Board:

```

1 #include <stdint.h>
2 #include <stdbool.h>
3 #include "inc/hw_types.h"
4 #include "inc/hw_memmap.h"
5 #include "driverlib/pin_map.h"
6
7 //This header File is important to Unlock GPIO Pins
8 #include "inc/hw_gpio.h"
9 #include "driverlib/sysctl.h"
10 #include "driverlib/gpio.h"
11 //Used for PWM
12 #include "driverlib/pwm.h"
13
14 #define right          0x22
15 #define left           0x11
16 #define softRight      0x10
17 #define softLeft       0x02
18 #define forward        0x12
19 #define backward       0x21
20 #define stop           0x00
21
22 void setupCLK();
23 void peripheralEnable();
24 void configIOPin();
25 void delay_ms(uint64_t delay);
26 void motion(uint8_t);
27 void enablePWM();
28 void Velocity(uint8_t lSpeed, uint8_t rSpeed);
29
30 int main(void) {
31     setupCLK();
32     peripheralEnable();
33     configIOPin();
34     enablePWM();
35     while(1{
36         Velocity(150, 150);
37         motion(forward);
38         delay_ms(2000);
39         motion(stop);
40         delay_ms(500);
41         Velocity(255, 255);
42         motion(backward);
43         delay_ms(800);
44         motion(stop);
45         delay_ms(500);
46         Velocity(255, 255);
47

```

```

48         motion(right);
49         delay_ms(1000);
50         motion(stop);
51         delay_ms(500);
52         Velocity(150, 150);
53         motion(left);
54         delay_ms(1000);
55         motion(stop);
56         delay_ms(500);
57         Velocity(150, 150);
58         motion(backward);
59         delay_ms(1000);
60     }
61 }
62 /**************************************************************************
63 * This function is used to setup Clock frequency of the controller
64 * It can be changed through codes
65 * In this we have set frequency as 40Mhz
66 * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
67 * * The PWM module is clocked by the system clock through a divider, and that divider has
68 * a range of 2 to 64.
69 * By setting the divider to 64, it will run the PWM clock at 625 kHz.
70 **************************************************************************/
71 void setupCLK(){
72     SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
73     SysCtlPWMClockSet(SYSCTL_PWMDIV_64); //625kHz PWM Clock
74 }
75 /**************************************************************************
76 * Enabling System Peripherals
77 * PORTF,PORTB and PORTA in this case
78 **************************************************************************/
79 void peripheralEnable(){
80     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
81     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
82     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
83     SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1); // Enabling PWM1
84 }
85 /**************************************************************************
86 * Configuring Pin as Input Or Output
87 * And Setting PWM Pin to Always High
88 **************************************************************************/
89 void configIOPin(){
90     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0|GPIO_PIN_1);
91     GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_5);
92     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_4);
93 }
94 /**************************************************************************
95 * Calculating Delays
96 **************************************************************************/
97 void delay_ms(uint64_t delay){
98     SysCtlDelay(delay*(SysCtlClockGet()/3000));
99 }
100 void delay_us(uint64_t delay){
101     SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
102 }
103 /**************************************************************************
104 * This function is for giving the direction of motion
105 * Macros have been defined at starting
106 * Macros for directions are 8 bits
107 * Out of these 8 bits only 4 are used
108 * Bit 0 corresponds to PB0
109 * Bit 1 corresponds to PB1
110 * Bit 4 corresponds to PF4
111 * Bit 5 corresponds to PA5
112 **************************************************************************/
113 void motion(uint8_t direction){
114     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1|GPIO_PIN_0, direction);
115     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_4, direction);
116     GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_5, direction);
117 }
118 /**************************************************************************
119 * This function is for enabling the PWM Modules
120 * PWM can be enabled on a pin based on the datasheet
121 **************************************************************************/
122 void enablePWM(){
123 }
```

```

124     GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_3);
125     GPIOPinConfigure(GPIO_PFF3_M1PWM7);
126     GPIOPinTypePWM(GPIO_PORTA_BASE, GPIO_PIN_6);
127     GPIOPinConfigure(GPIO_PA6_M1PWM2);
128 //Count Down Mode
129 PWMGenConfigure(PWM1_BASE, PWM_GEN_3, PWM_GEN_MODEDOWN | PWM_GEN_MODE_NOSYNC);
130 PWMGenPeriodSet(PWM1_BASE, PWM_GEN_3, 255); //Load Count value
131 //Count Down Mode
132 PWMGenConfigure(PWM1_BASE, PWM_GEN_1, PWM_GEN_MODEDOWN | PWM_GEN_MODE_NOSYNC);
133 PWMGenPeriodSet(PWM1_BASE, PWM_GEN_1, 255); //Load Count value
134 PWMGenEnable(PWM1_BASE, PWM_GEN_3); //Enable the generators
135 PWMGenEnable(PWM1_BASE, PWM_GEN_1);
136 PWMOOutputState(PWM1_BASE, PWM_OUT_7_BIT|PWM_OUT_2_BIT, true);
137 }
138 ****
139 * This function is used to control the speed of the motors
140 * The speed can changed by the PWMPulseWidthSet() function
141 * lSpeed is used to control the speed of left motor
142 * rSpeed is used to control the speed of right motor
143 ****
144 void Velocity(uint8_t lSpeed, uint8_t rSpeed){
145     lSpeed=(lSpeed>255)?255:lSpeed;
146     rSpeed=(rSpeed>255)?255:rSpeed;
147     PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7, lSpeed);
148     PWMPulseWidthSet(PWM1_BASE, PWM_OUT_2, rSpeed);
149 }
150

```

6.8 LCD Interfacing

To interface LCD with the microcontroller in default configuration requires 3 control signals and 8 data lines. This is known as 8 bit interfacing mode which requires total 11 I/O lines. To reduce the number of I/Os required for LCD interfacing we can use 4 bit interfacing mode which requires 3 control signals with 4 data lines. In this mode upper nibble and lower nibble of commands/data set needs to be sent separately. The three control lines are referred to as EN, RS, and RW. The LCD connections are given in section 5.6.

6.8.1 Code for Plug and Play Board:

```

1 #include <stdint.h>
2 #include <stdbool.h>
3 #include "inc/hw_types.h"
4 #include "inc/hw_memmap.h"
5 #include "inc/hw-gpio.h" //To unlock locked pins for GPIO
6 #include "driverlib/sysctl.h"
7 #include "driverlib/gpio.h"
8 #include <math.h>
9 #include <stdlib.h>

10
11 #ifndef lcdPORT
12 #define lcdPORT GPIO_PORTD_BASE
13 #endif
14 #ifndef lcdDDR
15 #define lcdDDR GPIO_PORTA_BASE
16 #endif
17 #ifndef lcdPIN
18 #define lcdPIN PINC
19 #endif
20 #ifndef RS
21 #define RS GPIO_PIN_6
22 #endif
23 #ifndef EN
24 #define EN GPIO_PIN_7
25 #endif
26 #ifndef D4
27 #define D4 GPIO_PIN_2
28 #endif
29 #ifndef D5
30 #define D5 GPIO_PIN_3
31 #endif
32 #ifndef D6
33

```

```

33     #define      D6          GPIO_PIN_4
34
35     #ifndef      D7          GPIO_PIN_5
36     #define      D7          GPIO_PIN_5
37     #endif
38     unsigned char cursorPositionCheck=0;
39
40     void lcdInit();
41     void lcdCommand(unsigned char);
42     void lcdData(unsigned char);
43     void lcdString(char* );
44     void lcdGotoxy(unsigned char ,unsigned char );
45     void lcdClear();
46     void lcdCheck();
47     void setupCLK();
48     void peripheralEnable();
49     void configIOPin();
50     void _delay_ms(uint64_t delay);
51     void _delay_us(uint64_t delay);
52
53     int main(){
54         setupCLK();
55         peripheralEnable();
56         configIOPin();
57         lcdInit();
58         lcdGotoxy(0,0);
59         lcdString("TIVA C Series");
60         while(1){
61     }
62 }
63 /*****
64 * This function is used to setup Clock frequency of the controller
65 * It can be changed through codes
66 * In this we have set frequency as 40Mhz
67 * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
68 *****/
69 void setupCLK(){
70     SysCtlClockSet(SYSCLOCK_SYSDIV_4|SYSCLOCK_USE_PLL|SYSCLOCK_XTAL_16MHZ|SYSCLOCK_OSC_MAIN);
71 }
72 *****
73 * Enabling System Peripherals
74 * PORTA and PORTD in this case
75 *****
76 void peripheralEnable(){
77     SysCtlPeripheralEnable(SYSCLOCK_PERIPH_GPIOA);
78     SysCtlPeripheralEnable(SYSCLOCK_PERIPH_GPIOB);
79 }
80 *****
81 * Configuring LCD Pins as Output
82 *****
83 void configIOPin(){
84     HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
85     HWREG(GPIO_PORTD_BASE + GPIO_O_CR) |= (1<<7);
86     HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = 0;
87     GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE,EN|RS);
88     GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE,D4|D5|D6|D7);
89 }
90 *****
91 * This function is used to initialize LCD
92 * The LCD is operated in 4 bit mode
93 * It is also set in auto increment mode
94 *****
95 void lcdInit(){
96     lcdCommand(0x28);
97     /*****
98     0x30 8bit mode single line*
99     0x38 8bit mode double line*
100    0x20 4bit mode single line*
101    0x28 4bit mode double line*
102    *****/
103    lcdCommand(0x06); //entry mode and auto increment mode
104    lcdCommand(0x0F);
105    /*****
106    Display off Cursor off      0x08*
107    Display on Cursor on       0x0E*
```

```

109     Display on Cursor off      0x0C*
110     Display on Cursor blinking 0x0F*
111     *****/
112 }
113 /*********************************************************************
114 * This function is used to send commands to LCD
115 * The first 4 bits are sent first and then the last 4 bits
116 * The bits are shifted left and right appropriately to fit in middle 4 bits of port A
117 *****/
118 void lcdCommand(unsigned char command){
119     GPIOPinWrite(lcdPORT,RS|EN,0) ;
120     GPIOPinWrite(lcdDDR,D4|D5|D6|D7,0) ;
121     _delay_us(100) ;
122     GPIOPinWrite(lcdDDR,D4|D5|D6|D7,( command>>2)) ;
123     _delay_ms(1) ;
124     GPIOPinWrite(lcdPORT,EN|RS,0x80) ;
125     _delay_us(100) ;
126     GPIOPinWrite(lcdPORT,EN,0) ;
127     _delay_us(100) ;
128     GPIOPinWrite(lcdDDR,D4|D5|D6|D7,( command<<2)) ;
129     _delay_ms(1) ;
130     GPIOPinWrite(lcdPORT,EN|RS,0x80) ;
131     _delay_us(100) ;
132     GPIOPinWrite(lcdPORT,EN,0) ;
133     _delay_us(100) ;
134 }
135 /*********************************************************************
136 * This function is used to send data to LCD
137 * The first 4 bits are sent first and then the last 4 bits
138 * The bits are shifted left and right appropriately to fit in middle 4 bits of port A
139 *****/
140 void lcdData(unsigned char data){
141     lcdCheck() ;
142     GPIOPinWrite(lcdPORT,RS|EN,0) ;
143     GPIOPinWrite(lcdDDR,D4|D5|D6|D7,0) ;
144     GPIOPinWrite(lcdDDR,D4|D5|D6|D7,( data>>2)) ;
145     _delay_us(100) ;
146     GPIOPinWrite(lcdPORT,EN|RS,0xc0) ;
147     _delay_ms(1) ;
148     GPIOPinWrite(lcdPORT,EN,0) ;
149     _delay_us(100) ;
150     GPIOPinWrite(lcdDDR,D4|D5|D6|D7,( data<<2)) ;
151     _delay_us(100) ;
152     GPIOPinWrite(lcdPORT,EN|RS,0xc0) ;
153     _delay_us(100) ;
154     GPIOPinWrite(lcdPORT,EN,0) ;
155     cursorPositionCheck=(cursorPositionCheck+1)%32;
156 }
157 /*********************************************************************
158 * This function is used to send strings to LCD
159 *****/
160 void lcdString(char* string){
161     unsigned char i=0;
162     while(string[i])
163         lcdData(string[i++]);
164 }
165 /*********************************************************************
166 * This function is used to position of the cursor on LCD
167 *****/
168 void lcdGotoxy(unsigned char x,unsigned char y)
169 {
170     cursorPositionCheck=y*16+x;
171     lcdCommand(0x80+x+(64*y));
172 }
173 /*********************************************************************
174 * This function is used to clear the LCD
175 *****/
176 void lcdClear(){
177     cursorPositionCheck=0;
178     lcdCommand(0x01);
179     _delay_ms(5) ;
180 }
181 /*********************************************************************
182 * This function is used to check the position of cursor on LCD
183 *****/
184 void lcdCheck(){

```

```

185     if(cursorPositionCheck==16)
186         lcdGotoxy(0,1);
187     else if(cursorPositionCheck==0)
188         lcdGotoxy(0,0);
189 }
190 /***** Calculating Delays *****/
191 * Calculating Delays
192 *****
193 void _delay_ms(uint64_t delay){
194     SysCtlDelay(delay*(SysCtlClockGet() /3000));
195 }
196 void _delay_us(uint64_t delay){
197     SysCtlDelay(delay*(SysCtlClockGet() /3000000UL));
198 }
199

```

6.8.2 Code for uC based Board

```

1 #include <stdint.h>
2 #include <stdbool.h>
3 #include "inc/hw_types.h"
4 #include "inc/hw_memmap.h"
5 #include "inc/hw_gpio.h" //To unlock locked pins for GPIO
6 #include "driverlib/sysctl.h"
7 #include "driverlib/gpio.h"
8 #include <math.h>
9 #include <stdlib.h>

10 #ifndef lcdPORT
11 #define lcdPORT    GPIO_PORTF_BASE
12 #endif
13
14 #ifndef lcdDDR
15 #define lcdDDR    GPIO_PORTD_BASE
16 #endif
17
18 #ifndef RS
19 #define RS        GPIO_PIN_0
20 #endif
21
22 #ifndef EN
23 #define EN        GPIO_PIN_2
24 #endif
25
26 #ifndef D4
27 #define D4        GPIO_PIN_4
28 #endif
29
30 #ifndef D5
31 #define D5        GPIO_PIN_5
32 #endif
33
34 #ifndef D6
35 #define D6        GPIO_PIN_6
36 #endif
37
38 #ifndef D7
39 #define D7        GPIO_PIN_7
40 #endif
41
42 #define LOCK_F (*((volatile unsigned long *)0x40025520))
43 #define CR_F   (*((volatile unsigned long *)0x40025524))
44
45 unsigned char cursorPositionCheck=0;
46
47 void lcdInit();
48 void lcdCommand(unsigned char);
49 void lcdData(unsigned char);
50 void lcdString(char*);
51 void lcdGotoxy(unsigned char,unsigned char);
52 void lcdClear();
53 void lcdCheck();
54 void setupCLK();
55 void peripheralEnable();
56 void configIOPin();
57 void _delay_ms(uint64_t delay);
58 void _delay_us(uint64_t delay);

59 int main(){
60     setupCLK();
61     peripheralEnable();
62     configIOPin();
63     lcdInit();
64     lcdClear();
65     lcdGotoxy(0,0);
66 }

```

```

59         lcdString("TIVA C Series");
60         while(1){
61     }
62     /*********************************************************************
63     * This function is used to setup Clock frequency of the controller
64     * It can be changed through codes
65     * In this we have set frequency as 40Mhz
66     * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
67     *********************************************************************/
68     void setupCLK(){
69         SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
70     }
71     /*********************************************************************
72     * Enabling System Peripherals
73     * PORTA and PORTD in this case
74     ********************************************************************/
75     void peripheralEnable(){
76         SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
77         SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
78     }
79     /*********************************************************************
80     * Configuring LCD Pins as Output
81     ********************************************************************/
82     void configIOPin(){
83         HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
84         HWREG(GPIO_PORTD_BASE + GPIO_O_CR) |= (1<<7);
85         HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = 0;
86
87         HWREG(GPIO_PORTE_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
88         HWREG(GPIO_PORTE_BASE + GPIO_O_CR) |= 0x01;
89         HWREG(GPIO_PORTE_BASE + GPIO_O_LOCK) = 0;
90         GPIOPinTypeGPIOOutput(lcdPORT,EN|RS);
91         GPIOPinTypeGPIOOutput(lcdDDR,D4|D5|D6|D7);
92     }
93     /*********************************************************************
94     * This function is used to initialize LCD
95     * The LCD is operated in 4 bit mode
96     * It is also set in auto increment mode
97     ********************************************************************/
98     void lcdInit(){
99         lcdCommand(0x28);
100        /*****
101        0x30 8bit mode single line*
102        0x38 8bit mode double line*
103        0x20 4bit mode single line*
104        0x28 4bit mode double line*
105        *****/
106        lcdCommand(0x06); //entry mode and auto increment mode
107        lcdCommand(0x0F); ///
108        /*****
109        Display off Cursor off    0x08*
110        Display on Cursor on    0x0E*
111        Display on Cursor off    0x0C*
112        Display on Cursor blinking 0x0F*
113        *****/
114    }
115    /*********************************************************************
116    * This function is used to send commands to LCD
117    * The first 4 bits are sent first and then the last 4 bits
118    * The bits are shifted left to fit in middle 4 bits of port D
119    ********************************************************************/
120    void lcdCommand(unsigned char command){
121        GPIOPinWrite(lcdPORT,RS|EN,0);
122        GPIOPinWrite(lcdDDR,D4|D5|D6|D7,0);
123        _delay_us(100);
124        GPIOPinWrite(lcdDDR,D4|D5|D6|D7,command);
125        _delay_us(100);
126        GPIOPinWrite(lcdPORT,EN|RS,0x04);
127        _delay_ms(1);
128        GPIOPinWrite(lcdPORT,EN,0);
129        _delay_us(100);
130        GPIOPinWrite(lcdDDR,D4|D5|D6|D7,(command<<4));
131        _delay_ms(1);
132        GPIOPinWrite(lcdPORT,EN|RS,0x04);
133        _delay_us(100);
134    }

```

```

135     GPIOPinWrite(lcdPORT,EN,0) ;
136     _delay_us(100) ;
137 }
138 /**
139 * This function is used to send data to LCD
140 * The first 4 bits are sent first and then the last 4 bits
141 * The bits are shifted left to fit in middle 4 bits of port D
142 */
143 void lcdData(unsigned char data){
144     lcdCheck() ;
145     GPIOPinWrite(lcdPORT,RS|EN,0) ;
146     GPIOPinWrite(lcdDDR,D4|D5|D6|D7,0) ;
147     GPIOPinWrite(lcdDDR,D4|D5|D6|D7,data) ;
148     _delay_us(100) ;
149     GPIOPinWrite(lcdPORT,EN|RS,0x05) ;
150     _delay_ms(1) ;
151     GPIOPinWrite(lcdPORT,EN,0) ;
152     _delay_us(100) ;
153     GPIOPinWrite(lcdDDR,D4|D5|D6|D7,(data<<4)) ;
154     _delay_us(100) ;
155     GPIOPinWrite(lcdPORT,EN|RS,0x05) ;
156     _delay_ms(1) ;
157     GPIOPinWrite(lcdPORT,EN,0) ;
158     cursorPositionCheck=(cursorPositionCheck+1)%32;
159 }
160 /**
161 * This function is used to send strings to LCD
162 */
163 void lcdString(char* string){
164     unsigned char i=0;
165     while(string[i])
166         lcdData(string[i++]);
167 }
168 /**
169 * This function is used to position of the cursor on LCD
170 */
171 void lcdGotoxy(unsigned char x,unsigned char y)
172 {
173     cursorPositionCheck=y*16+x;
174     lcdCommand(0x80+x+(64*y));
175 }
176 /**
177 * This function is used to clear the LCD
178 */
179 void lcdClear(){
180     cursorPositionCheck=0;
181     lcdCommand(0x01);
182     _delay_ms(3);
183 }
184 /**
185 * This function is used to check the position of cursor on LCD
186 */
187 void lcdCheck(){
188     if(cursorPositionCheck==16)
189         lcdGotoxy(0,1);
190     else if(cursorPositionCheck==0)
191         lcdGotoxy(0,0);
192 }
193 /**
194 * Calculating Delays
195 */
196 void _delay_ms(uint64_t delay){
197     SysCtlDelay(delay*(SysCtlClockGet()/3000));
198 }
199 void _delay_us(uint64_t delay){
200     SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
201 }
202

```

6.9 Analog To Digital Converter

Fire Bird V has three white line sensors, one Sharp IR range sensor with four add-on sockets for additional Sharp IR range sensors, eight Analog IR proximity sensors. All these sensors give analog output. We need to use ADC (Analog to Digital Converter) to convert these analog values in to digital values.

The TM4C123GH6PM ADC module features 12-bit conversion resolution and supports 12 input channels, plus an internal temperature sensor. Each ADC module contains four programmable sequencers allowing the sampling of multiple analog input sources without controller intervention. Each sample sequencer provides flexible programming with fully configurable input source, trigger events, interrupt generation, and sequencer priority.

Due to limited number of sensors in TM4C123GH6PM, an external ADC(ADC128d818) is added to the daughter Board. Details about interfacing this module is given in the next section.

The Connections of internal ADC is as shown below

Plug and Play Board			uC Based Board	
ADC Channel no.	Port	Sensor	Port	Sensor
0	PE3	White line sensor 1	PE3	IR Proximity analog sensor 2
1	PE2	White line sensor 2	PE2	Sharp IR range sensor 2
2	PE1	White line sensor 3	PE1	IR Proximity analog sensor 1
3	PE0	IR Proximity analog sensor 7	PE0	Sharp IR range sensor 1
4	PD3	IR Proximity analog sensor 3	PD3	Sharp IR range sensor 3
5	PD2	Sharp IR range sensor 3	PD2	White line sensor 3
6	PD1	IR Proximity analog sensor 4	PD1	White line sensor 2
7	PDO	IR Proximity analog sensor 2	PDO	White line sensor 1
8	PE5	IR Proximity analog sensor 5	PE5	IR Proximity analog sensor 3
9	PE4	Sharp IR range sensor 5	PE4	IR Proximity analog sensor 4
10	PB4	Sharp IR range sensor 1	PB4	Sharp IR range sensor 5
11	PB5	IR Proximity analog sensor 1	PB5	IR Proximity analog sensor 5

6.9.1 Code for Plug and Play Board:

```

1 #include <stdint.h>
2 #include <stdbool.h>
3 #include "stdlib.h"
4 #include "inc/hw_ints.h"
5 #include "inc/hw_memmap.h"
6 #include "inc/hw_uart.h"
7 #include "inc/hw_gpio.h"
8 #include "inc/hw_pwm.h"
9 #include "inc/hw_types.h"
10 #include "driverlib/adc.h"
11 #include "driverlib/timer.h"
12 #include "driverlib/gpio.h"
13 #include "driverlib/interrupt.h"
14 #include "driverlib/pin_map.h"
15 #include "driverlib/rom.h"
16 #include "driverlib/rom_map.h"
17 #include "driverlib/sysctl.h"
18 #include "driverlib/uart.h"
19 #include "driverlib/udma.h"
20 #include "driverlib/pwm.h"
21 #include "driverlib/ssi.h"
22 #include "driverlib/systick.h"
23 #include "driverlib/adc.h"
24 #include "utils/uartstdio.h"
25 #include "utils/uartstdio.c"
26 #include <string.h>

27 void configCLK();
28 void peripheralEnable();
29 void uartEnable();

30 void ADC0Enable();
31 unsigned int readADC();
32 void tranString(char * data, char delimiter);
33 void uartInteger(long long int integer, char delimiter);
34 void converter(unsigned int);
35
36

```

```

37     void _delay_ms(uint64_t delay);
38     uint32_t senval;
39
40     int main(){
41         configCLK();
42         peripheralEnable();
43         ADC0Enable();
44         uartEnable();
45         while(1){
46             senval = readADC();
47             converter(senval);
48             _delay_ms(1000);
49         }
50     }
51 /**
52 * This function is used to setup Clock frequency of the controller
53 * It can be changed through codes
54 * In this we have set frequency as 40Mhz
55 * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
56 */
57 void configCLK(){
58     SysCtlClockSet(SYSCLOCK_SYSDIV_5|SYSCLOCK_USE_PLL|SYSCLOCK_OSC_MAIN|SYSCLOCK_XTAL_16MHZ);
59 }
60 /**
61 * Enabling System Peripherals
62 * PortB and PortD in this case
63 */
64 void peripheralEnable(){
65     SysCtlPeripheralEnable(SYSCLOCK_PERIPH_UART1);
66     SysCtlPeripheralEnable(SYSCLOCK_PERIPH_GPIOB); //Enabling TIMER0
67     SysCtlPeripheralEnable(SYSCLOCK_PERIPH_GPIOD);
68     SysCtlPeripheralEnable(SYSCLOCK_PERIPH_ADC0);
69     ADCHardwareOversampleConfigure(ADC0_BASE, 64);
70 }
71 /**
72 * This function is used to enable UART1
73 * The baudrate is set at 9600
74 */
75 void uartEnable(){
76     GPIOPinConfigure(GPIO_PB0_U1RX); //Configure Pin B0 as RX of U0
77     GPIOPinConfigure(GPIO_PB1_U1TX); //Configure Pin B1 as TX of U0
78     GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);
79     UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 9600,(UART_CONFIG_WLEN_8 |
80     UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
81 }
82 /**
83 * This function is used to enable ADC0
84 * 4 step sequencer is used
85 * Change the channel number to use any of the other ADCs
86 */
87 void ADC0Enable(){
88     ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
89     ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_CH4);
90     ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_CH4);
91     ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_CH4);
92     ADCSequenceStepConfigure(ADC0_BASE, 1, 3, ADC_CTL_CH4|ADC_CTL_IE|ADC_CTL_END);
93     ADCSequenceEnable(ADC0_BASE, 1);
94     GPIOPinTypeADC(GPIO_PORTD_BASE, GPIO_PIN_3);
95 }
96 /**
97 * This function is used to read the value from ADC
98 * Average of 4 values is returned to the calling function
99 */
100 unsigned int readADC(){
101     unsigned int Avg;
102     uint32_t ADC0Value[4];
103     ADCIntClear(ADC0_BASE, 1);
104     ADCProcessorTrigger(ADC0_BASE, 1);
105     while(!ADCIntStatus(ADC0_BASE, 1, false));
106     ADCSequenceDataGet(ADC0_BASE, 1, ADC0Value);
107     Avg = (ADC0Value[0] + ADC0Value[1] + ADC0Value[2] + ADC0Value[3] + 2)/4;
108     return(Avg);
109 }
110 /**
111 * This function is used to send the ADC values through UART
112 * Here the value is sent in reverse order

```

```

112     *****/
113     void converter(uint32_t q)
114     {
115         unsigned int p;
116         p=q;
117         do
118         {
119             p = (q % 10);
120             UARTCharPut(UART1_BASE,48+(int)p);
121             SysCtlDelay(400000);
122             q = q / 10;
123         }while(q != 0);
124         UARTCharPut(UART1_BASE, ' ');
125     }
126     *****/
127     * Calculating Delays
128     *****/
129     void _delay_ms(uint64_t delay){
130         SysCtlDelay(delay*(SysCtlClockGet() /3000));
131     }
132

```

6.9.2 Code for uC based Board

```

1 #include <stdint.h>
2 #include <stdbool.h>
3 #include "stdlib.h"
4 #include "inc/hw_ints.h"
5 #include "inc/hw_memmap.h"
6 #include "inc/hw_uart.h"
7 #include "inc/hw_gpio.h"
8 #include "inc/hw_pwm.h"
9 #include "inc/hw_types.h"
10 #include "driverlib/adc.h"
11 #include "driverlib/timer.h"
12 #include "driverlib/gpio.h"
13 #include "driverlib/interrupt.h"
14 #include "driverlib/pin_map.h"
15 #include "driverlib/rom.h"
16 #include "driverlib/rom_map.h"
17 #include "driverlib/sysctl.h"
18 #include "driverlib/uart.h"
19 #include "driverlib/udma.h"
20 #include "driverlib/pwm.h"
21 #include "driverlib/ssi.h"
22 #include "driverlib/systick.h"
23 #include "driverlib/adc.h"
24 #include "utils/uartstdio.h"
25 #include "utils/uartstdio.c"
26 #include <string.h>
27 #include <math.h>
28
29 void configCLK();
30 void peripheralEnable();
31 void uartEnable();
32 unsigned int Sharp_GP2D12_estimation(uint16_t adc_reading);
33 void ADC0Enable();
34 unsigned int readADC();
35 void tranString(char * data,char delimiter);
36 void uartInteger(long long int integer,char delimiter);
37 void _delay_ms(uint64_t delay);
38 void itoa(long long a,char *arr);
39
40 int main(){
41     configCLK();
42     peripheralEnable();
43     ADC0Enable();
44     uartEnable();
45
46     while(1){
47         uartInteger(Sharp_GP2D12_estimation(readADC()),' ');
48         _delay_ms(1000);
49     }
50 }
51 *****/
52 * This function is used to setup Clock frequency of the controller

```

```

53 * It can be changed through codes
54 * In this we have set frequency as 40Mhz
55 * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
56 *****/
57 void configCLK(){
58     SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
59 }
60 *****/
61 * Enabling System Peripherals
62 * PortB and PortD in this case
63 *****/
64 void peripheralEnable(){
65     SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
66     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
67     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
68     SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
69     ADCHardwareOversampleConfigure(ADC0_BASE, 64);
70 }
71 *****/
72 * This function is used to enable UART1
73 * The baudrate is set at 9600
74 *****/
75 void uartEnable(){
76     GPIOPinConfigure(GPIO_PC4_U1RX); //Configure Pin B0 as RX of U0
77     GPIOPinConfigure(GPIO_PC5_U1TX); //Configure Pin B1 as TX of U0
78     GPIOPinTypeUART(GPIO_PORTC_BASE, GPIO_PIN_5 | GPIO_PIN_4);
79     UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 9600,(UART_CONFIG_WLEN_8 |
    UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
80 }
81 *****/
82 * This function is used to enable ADC0
83 * 4 step sequencer is used
84 * Change the channel number to use any of the other ADCs
85 *****/
86 void ADC0Enable(){
87     ADCSequenceConfigure(ADC0.BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
88     ADCSequenceStepConfigure(ADC0.BASE, 1, 0, ADC_CTL_CH1);
89     ADCSequenceStepConfigure(ADC0.BASE, 1, 1, ADC_CTL_CH1);
90     ADCSequenceStepConfigure(ADC0.BASE, 1, 2, ADC_CTL_CH1);
91     ADCSequenceStepConfigure(ADC0.BASE, 1, 3, ADC_CTL_CH1|ADC_CTL_IE|ADC_CTL_END);
92     ADCSequenceEnable(ADC0.BASE, 1);
93     GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_2);
94 }
95 *****/
96 * This function is used to read the value from ADC
97 * Average of 4 values is returned to the calling function
98 *****/
99 unsigned int readADC(){
100     unsigned int Avg;
101     uint32_t ADC0Value[4];
102     ADCIntClear(ADC0.BASE, 1);
103     ADCProcessorTrigger(ADC0.BASE, 1);
104     while(!ADCIntStatus(ADC0.BASE, 1, false));
105     ADCSequenceDataGet(ADC0.BASE, 1, ADC0Value);
106     Avg = (ADC0Value[0] + ADC0Value[1] + ADC0Value[2] + ADC0Value[3] + 2) / 4;
107     return(Avg);
108 }
109 void itoa(long long a,char *arr){
110     int i=0,j=0;
111     long long tmp=a;
112     if(a<0){
113         arr[i++]= '-';
114         tmp*=-1;
115         j=1;
116     }
117     for(;tmp>0;i++){
118         arr[i]=(tmp%10)+'0';
119         tmp/=10;
120     }
121     arr[i--]='\0';
122     for(;j<i;j++,i--){
123         tmp=arr[i];
124         arr[i]=arr[j];
125         arr[j]=tmp;
126     }
127 }
```

```

128  ****
129 * Calculating Delays
130 ****
131 void _delay_ms(uint64_t delay){
132     SysCtlDelay(delay*(SysCtlClockGet() / 3000));
133 }
134 void uartInteger(long long int integer, char delimiter){
135     char ch[20];
136     itoa(integer, ch);
137     tranString(ch, delimiter);
138 }
139 void tranString(char *data, char delimiter){
140     int k=0;
141     while(data[k]){
142         UARTCharPut(UART1_BASE, data[k++]);
143     }
144     UARTCharPut(UART1_BASE, delimiter);
145 }
146 unsigned int Sharp_GP2D12_estimation(uint16_t adc_reading){
147     float distance;
148     unsigned int distanceInt;
149     distance = (int)(10.00*(2799.6*(1.00/(pow(adc_reading ,1.1546))))));
150     distanceInt = (int)distance;
151     if(distanceInt >800){
152         distanceInt=800;
153     }
154     return distanceInt;
155 }
156

```

6.10 External ADC Interface

The TM4C123GHP6 microcontroller has only 12 internal ADC's. So to interface all the sensors of Firebird an external ADC is required. For this purpose *adc128d818* IC is used. It has 12 bit resolution with 8 channels. It can be interfaced using I2C protocols. For more details about the click on this link [adc128d818](#)

6.10.1 Code for Plug and Play Board:

```

1 #include <stdarg.h>
2 #include <stdbool.h>
3 #include <stdint.h>
4 #include "inc/hw_i2c.h"
5 #include "inc/hw_memmap.h"
6 #include "inc/hw_types.h"
7 #include "inc/hw_gpio.h"
8 #include "driverlib/i2c.h"
9 #include "driverlib/sysctl.h"
10 #include "driverlib/gpio.h"
11 #include "driverlib/pin_map.h"
12 #include "inc/tm4c123gh6pm.h"
13 #include "driverlib/interrupt.h"
14 #include "driverlib/uart.h"
15
16 void setupCLK();
17 void InitI2C1(void);
18 void peripheralEnable();
19 void I2CSendString(uint32_t slave_addr, char array[]);
20 void I2CSend(uint8_t slave_addr, uint8_t num_of_args, ...);
21 uint32_t I2CReceive(uint32_t slave_addr, uint8_t reg);
22 uint16_t read2Bytes(uint8_t slave_addr, uint8_t reg);
23 uint16_t readExternalADC(uint8_t channel);
24 void uartEnable();
25 void uartInterruptEnable();
26 void UARTIntHandler(void);
27 void tranString(char *,char);
28 void itoa(long long a,char *arr);
29 void uartInteger(long long int integer,char delimiter);
30 void setupExternalADC();
31
32 int main(void) {
33     setupCLK();
34     peripheralEnable();
35     uartEnable();

```

```

36     InitI2C1();
37     setupExternalADC();
38     while(1){
39         uartInteger(readExternalADC(0), ',');
40         SysCtlDelay((SysCtlClockGet())/3);
41     }
42 }
43
44 /***** This function is used to setup Clock frequency of the controller ****
45 * It can be changed through codes
46 * In this we have set frequency as 40Mhz
47 * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
48 ****
49 void setupCLK(){
50     SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
51 }
52 ****
53 * Enabling System Peripherals
54 * UART1 and PortB in this case
55 ****
56 void peripheralEnable(){
57     SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
58     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
59 }
60 ****
61 * This function is used to enable UART0
62 * The baudrate is set at 9600
63 ****
64 void uartEnable(){
65     GPIOPinConfigure(GPIO_PB0_U1RX);
66     GPIOPinConfigure(GPIO_PB1_U1TX);
67     GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);
68     UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 9600,
69     (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
70 }
71
72 ****
73 * This function converts an integer to string
74 ****
75 void itoa(long long a,char *arr){
76     int i=0,j=0;
77     long long tmp=a;
78     if(a<0){
79         arr [i++]= '-';
80         tmp*=-1;
81         j=1;
82     }
83     for (;tmp>0;i++){
84         arr [i]=(tmp%10)+'0';
85         tmp/=10;
86     }
87     arr [i--]='\0';
88     for (;j<i;j++,i--){
89         tmp=arr [i];
90         arr [i]=arr [j];
91         arr [j]=tmp;
92     }
93 }
94
95 ****
96 * send through serial communication
97 ****
98 void uartInteger(long long int integer,char delimiter){
99     char ch[20];
100    itoa(integer, ch);
101    tranString(ch, delimiter);
102 }
103 ****
104 * This function is used to send a string of character with a delimiter
105 ****
106 void tranString(char * data,char delimiter){
107     int k=0;
108     while(data[k]){
109         UARTCharPut(UART1_BASE, data [k++]);
110     }

```

```

112     UARTCharPut(UART1_BASE, delimiter);
113 }
114
115 /* *****
116 * Setting up external ADC
117 * Wait for 33ms
118 * Read the Busy Status Register (address 0Ch). If the 'Not Ready' * * bit = 1,
119 * then increase the wait time until 'Not Ready' bit = 0 before * proceeding to the next step
120
121 * Program the Advanced Configuration Register to Address 0Bh:
122 * a. Choose to use the internal or external VREF (bit 0).
123 * b. Choose the mode of operation (bits [2:1]).
124 * Program the Conversion Rate Register (address 07h).
125 * Choose to enable or disable the channels using the Channel Disable Register (address 08h).
126 * Using the Interrupt Mask Register (address 03h), choose to mask *or not to mask the interrupt
status from
127 *propagating to the interrupt output pin, INT.
128 * Program the Limit Registers (addresses 2Ah to 39h).
129 * Set the START bit of the Configuration Register (address 00h, bit 0) to 1.
130 * Set the 'INT_Clear' bit (address 00h, bit 3) to 0. If needed, program the 'INT_Enable' bit (address 00h, bit 1)
131 to 1 to enable the INT output.
132 *****/
133 void setupExternalADC(){
134     SysCtlDelay((5*SysCtlClockGet())/300);
135     while(I2CReceive(0x1D,0x0C)&(0x01));
136     I2CSend(0x1D,2,0x0B,0x03);
137     I2CSend(0x1D,2,0x07,0x01);
138     I2CSend(0x1D,2,0x08,0x00);
139     I2CSend(0x1D,3,0x2A,0x05);
140     I2CSend(0x1D,3,0x2B,0x00);
141     I2CSend(0x1D,3,0x2C,0x05);
142     I2CSend(0x1D,3,0x2D,0x00);
143     I2CSend(0x1D,3,0x2E,0x05);
144     I2CSend(0x1D,3,0x2F,0x00);
145     I2CSend(0x1D,3,0x30,0x05);
146     I2CSend(0x1D,3,0x31,0x00);
147     I2CSend(0x1D,3,0x32,0x05);
148     I2CSend(0x1D,3,0x33,0x00);
149     I2CSend(0x1D,3,0x34,0x05);
150     I2CSend(0x1D,3,0x35,0x00);
151     I2CSend(0x1D,3,0x36,0x05);
152     I2CSend(0x1D,3,0x37,0x00);
153     I2CSend(0x1D,3,0x38,0x05);
154     I2CSend(0x1D,3,0x39,0x00);
155     I2CSend(0x1D,2,0x00,0x01);
156 }
157 /* *****
158 * Read ADC value from a mentioned channel
159 *****/
160 uint16_t readExternalADC(uint8_t channel){
161     while(I2CReceive(0x1D,0x0C)&(0x02));
162     return(read2Bytes(0x1D,(0x20+channel)));
163 }
164 /* *****
165 * This function is used to enable I2C1
166 *****/
167 void InitI2C1(void){
168     SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C1);
169     SysCtlPeripheralReset(SYSCTL_PERIPH_I2C1);
170     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
171     GPIOPinConfigure(GPIO_PA6_I2C1SCL);
172     GPIOPinConfigure(GPIO_PA7_I2C1SDA);
173     GPIOPinTypeI2CSCL(GPIO_PORTA_BASE, GPIO_PIN_6);
174     GPIOPinTypeI2C(GPIO_PORTA_BASE, GPIO_PIN_7);
175     // Enable and initialize the I2C1 master module. Use the system clock for
176     // the I2C1 module. The last parameter sets the I2C data transfer rate.
177     // If false the data rate is set to 100kbps and if true the data rate will
178     // be set to 400kbps.
179     I2CMasterInitExpClk(I2C1_BASE, SysCtlClockGet(), false);
180     HWREG(I2C1_BASE + I2C_O_FIFOTL) = 80008000;
181 }
182 /* *****
183 * This function is used to send n number of data on serial Data line
184 *****/
185 void I2CSend(uint8_t slave_addr, uint8_t num_of_args, ...){

```

```

185     I2CMasterSlaveAddrSet(I2C1_BASE, slave_addr, false);
186     va_list vargs;
187     va_start(vargs, num_of_args);
188     I2CMasterDataPut(I2C1_BASE, va_arg(vargs, uint32_t));
189     if (num_of_args == 1){
190         I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_SEND);
191         while (I2CMasterBusy(I2C1_BASE));
192         va_end(vargs);
193     }
194     else{
195         I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_START);
196         while (I2CMasterBusy(I2C1_BASE));
197         uint8_t i ;
198         for(i = 1; i < (num_of_args - 1); i++){
199             I2CMasterDataPut(I2C1_BASE, va_arg(vargs, uint32_t));
200             I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);
201             while (I2CMasterBusy(I2C1_BASE));
202         }
203         I2CMasterDataPut(I2C1_BASE, va_arg(vargs, uint32_t));
204         I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
205         while (I2CMasterBusy(I2C1_BASE));
206         va_end(vargs);
207     }
208 }
209
210 /*********************************************************************
211 * This function is used to send string data on serial Data line
212 *****/
213 void I2CSendString(uint32_t slave_addr, char array[]){
214     I2CMasterSlaveAddrSet(I2C1_BASE, slave_addr, false);
215     I2CMasterDataPut(I2C1_BASE, array[0]);
216     if(array[1] == '\0'){
217         I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_SEND);
218         while (I2CMasterBusy(I2C1_BASE));
219     }
220     else{
221         I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_START);
222         while (I2CMasterBusy(I2C1_BASE));
223         uint8_t i = 1;
224         while(array[i + 1] != '\0'){
225             I2CMasterDataPut(I2C1_BASE, array[i++]);
226             I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);
227             while (I2CMasterBusy(I2C1.BASE));
228         }
229         I2CMasterDataPut(I2C1_BASE, array[i]);
230         I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
231         while (I2CMasterBusy(I2C1.BASE));
232     }
233 }
234
235 /*********************************************************************
236 * This function is used to receive 1 data on serial Data line
237 *****/
238 uint32_t I2CReceive(uint32_t slave_addr, uint8_t reg){
239     I2CMasterSlaveAddrSet(I2C1_BASE, slave_addr, false);
240     I2CMasterDataPut(I2C1_BASE, reg);
241     I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_START);
242     while (I2CMasterBusy(I2C1_BASE));
243     I2CMasterSlaveAddrSet(I2C1_BASE, slave_addr, true);
244     I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
245     while (I2CMasterBusy(I2C1.BASE));
246     return I2CMasterDataGet(I2C1.BASE); //return data pulled from the specified register
247 }
248
249 /*********************************************************************
250 * This function is used to receive 2 data on serial Data line
251 *****/
252 uint16_t read2Bytes(uint8_t SLAVE_ADDRESS, uint8_t reg){
253     I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, false);
254     I2CMasterDataPut(I2C1_BASE, reg);
255     I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_START);
256     while (!(I2CMasterBusy(I2C1.BASE))) {};
257     while (I2CMasterBusy(I2C1.BASE)) {};
258     I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, true);
259     I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START);
260     while (!(I2CMasterBusy(I2C1.BASE))) {};
261     while (I2CMasterBusy(I2C1.BASE)) {};

```

```

261     uint8_t data=I2CMasterDataGet ( I2C1_BASE );
262     I2CMasterControl(I2C1_BASE , I2C_MASTER_CMD_BURST_RECEIVE_FINISH) ;
263     while (! ( I2CMasterBusy (I2C1_BASE) )) {};
264     while (I2CMasterBusy (I2C1_BASE)) {};
265     uint8_t dummy=I2CMasterDataGet ( I2C1_BASE );
266     return (((((data<<8)+dummy )>>4);
267 }
268

```

6.10.2 Code for uC based Board:

```

1 #include <stdarg.h>
2 #include <stdbool.h>
3 #include <stdint.h>
4 #include "inc/hw_i2c.h"
5 #include "inc/hw_memmap.h"
6 #include "inc/hw_types.h"
7 #include "inc/hw_gpio.h"
8 #include "driverlib/i2c.h"
9 #include "driverlib/sysctl.h"
10 #include "driverlib/gpio.h"
11 #include "driverlib/pin_map.h"
12 #include "inc/tm4c123gh6pm.h"
13 #include "driverlib/interrupt.h"
14 #include "driverlib/uart.h"
15
16 void setupCLK();
17 void InitI2C0(void);
18 void peripheralEnable();
19 void I2CSendString(uint32_t slave_addr, char array []);
20 void I2CSend(uint8_t slave_addr, uint8_t num_of_args, ...);
21 uint32_t I2CReceive(uint32_t slave_addr, uint8_t reg);
22 uint16_t read2Bytes(uint8_t slave_addr, uint8_t reg);
23 uint16_t readExternalADC(uint8_t channel);
24 void uartEnable();
25 void transString(char *,char);
26 void itoa(long long a,char *arr);
27 void uartInteger(long long int integer,char delimiter);
28 void setupExternalADC();
29
30 int main(void) {
31     setupCLK();
32     peripheralEnable();
33     uartEnable();
34     transString("Ayush", ' ');
35     InitI2C0();
36     setupExternalADC();
37     uint16_t val0=0;
38     while(1){
39         val0=readExternalADC(5);
40         uartInteger(readExternalADC(0), ' ');
41         SysCtlDelay((SysCtlClockGet()) / 3);
42     }
43 }
44
45 /*********************************************************************
46 * This function is used to setup Clock frequency of the controller
47 * It can be changed through codes
48 * In this we have set frequency as 40Mhz
49 * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
50 *********************************************************************/
51 void setupCLK(){
52     SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
53 }
54 /*********************************************************************
55 * Enabling System Peripherals
56 * UART1 and PortC in this case
57 *****************************************************************/
58 void peripheralEnable(){
59     SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
60     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
61 }
62 /*********************************************************************
63 * This function is used to enable UART1
64 * The baudrate is set at 9600
65

```

```

66 *****/
67 void uartEnable(){
68   GPIOPinConfigure(GPIO_PC4_U1RX); //Configure Pin A0 as RX of U0
69   GPIOPinConfigure(GPIO_PC5_U1TX); //Configure Pin A1 as TX of U0
70   GPIOPinTypeUART(GPIO_PORTC_BASE, GPIO_PIN_4 | GPIO_PIN_5);
71   UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 9600,
72   (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
73 }
74 *****/
75 * This function converts an integer to string
76 *****/
77 void itoa(long long a,char *arr){
78   int i=0,j=0;
79   long long tmp=a;
80   if(a<0){
81     arr [ i++]= '-';
82     tmp*=-1;
83     j=1;
84   }
85   for (;tmp>0;i++){
86     arr [ i]=(tmp%10)+ '0';
87     tmp/=10;
88   }
89   arr [ i--]='\0';
90   for (;j<i;j++,i--){
91     tmp=arr [ i];
92     arr [ i]=arr [ j];
93     arr [ j]=tmp;
94   }
95 }
96 *****/
97 * send through serial communication
98 *****/
99 void uartInteger(long long int integer ,char delimiter){
100   char ch[20];
101   itoa(integer ,ch);
102   tranString(ch ,delimiter);
103 }
104 *****/
105 * This function is used to send a string of character with a delimiter
106 *****/
107 void tranString(char * data ,char delimiter){
108   int k=0;
109   while(data[k]){
110     UARTCharPut(UART1_BASE, data [ k++]);
111   }
112   UARTCharPut(UART1_BASE, delimiter);
113 }
114 *****/
115 * Setting up external ADC
116 * Wait for 33ms
117 * Read the Busy Status Register (address 0Ch). If the 'Not Ready' bit = 1,
118 * then increase the wait time until 'Not Ready' bit = 0 before proceeding to the next step.
119 * Program the Advanced Configuration Register Address 0B\h:
120 * a. Choose to use the internal or external VREF (bit 0).
121 * b. Choose the mode of operation (bits [2:1]).
122 * Program the Conversion Rate Register (address 07h).
123 * Choose to enable or disable the channels using the Channel Disable Register (address 08h).
124 * Using the Interrupt Mask Register (address 03h), choose to mask or not to mask the interrupt
125 status from
126 propagating to the interrupt output pin, INT.
127 * Program the Limit Registers (addresses 2Ah to 39h).
128 * Set the START bit of the Configuration Register (address 00h, bit 0) to 1.
129 * Set the 'INT_Clear' bit (address 00h, bit 3) to 0. If needed, program the 'INT_Enable' bit (
130 address 00h, bit 1)
131 to 1 to enable the INT output.
132 *****/
133 void setupExternalADC(){
134   SysCtlDelay((5*SysCtlClockGet()) /300);
135   while(I2CReceive(0x1D,0x0C)&(0x01));
136   I2CSend(0x1D,2,0x0B,0x03);
137   I2CSend(0x1D,2,0x07,0x01);
138   I2CSend(0x1D,2,0x08,0x00);
139   I2CSend(0x1D,3,0x2A,0x05);

```

```

140     I2CSend(0x1D,3,0x2B,0x00);
141     I2CSend(0x1D,3,0x2C,0x05);
142     I2CSend(0x1D,3,0x2D,0x00);
143     I2CSend(0x1D,3,0x2E,0x05);
144     I2CSend(0x1D,3,0x2F,0x00);
145     I2CSend(0x1D,3,0x30,0x05);
146     I2CSend(0x1D,3,0x31,0x00);
147     I2CSend(0x1D,3,0x32,0x05);
148     I2CSend(0x1D,3,0x33,0x00);
149     I2CSend(0x1D,3,0x34,0x05);
150     I2CSend(0x1D,3,0x35,0x00);
151     I2CSend(0x1D,3,0x36,0x05);
152     I2CSend(0x1D,3,0x37,0x00);
153     I2CSend(0x1D,3,0x38,0x05);
154     I2CSend(0x1D,3,0x39,0x00);
155     I2CSend(0x1D,2,0x00,0x01);
156 }
157 /**
158 * Read ADC value from a mentioned channel
159 */
160 uint16_t readExternalADC(uint8_t channel){
161     while(I2CReceive(0x1D,0x0C)&(0x02));
162     return(read2Bytes(0x1D,(0x20+channel)));
163 }
164 /**
165 * This function is used to enable I2C0
166 */
167 void InitI2C0(void){
168     SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);
169     SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);
170     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
171     GPIOPinConfigure(GPIO_PB2_I2C0SCL);
172     GPIOPinConfigure(GPIO_PB3_I2C0SDA);
173     GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
174     GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);
175     // Enable and initialize the I2C0 master module. Use the system clock for
176     // the I2C0 module. The last parameter sets the I2C data transfer rate.
177     // If false the data rate is set to 100kbps and if true the data rate will
178     // be set to 400kbps.
179     I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), false);
180     HWREG(I2C0_BASE + I2C_O_FIFOCNT) = 80008000;
181 }
182 /**
183 * This function is used to send n number of data on serial Data line
184 */
185 void I2CSend(uint8_t slave_addr, uint8_t num_of_args, ...){
186     I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);
187     va_list vargs;
188     va_start(vargs, num_of_args);
189     I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));
190     if(num_of_args == 1){
191         I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);
192         while(I2CMasterBusy(I2C0_BASE));
193         va_end(vargs);
194     }
195     else{
196         I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
197         while(I2CMasterBusy(I2C0_BASE));
198         uint8_t i ;
199         for(i = 1; i < (num_of_args - 1); i++){
200             I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));
201             I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);
202             while(I2CMasterBusy(I2C0_BASE));
203         }
204         I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));
205         I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
206         while(I2CMasterBusy(I2C0_BASE));
207         va_end(vargs);
208     }
209 }
210 /**
211 * This function is used to send string data on serial Data line
212 */
213 void I2CSendString(uint32_t slave_addr, char array[]){
214     I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);

```

```

216     I2CMasterDataPut(I2C0_BASE, array[0]);
217     if(array[1] == '\0')
218     {
219         I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);
220         while(I2CMasterBusy(I2C0_BASE));
221     }
222     else{
223         I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
224         while(I2CMasterBusy(I2C0_BASE));
225         uint8_t i = 1;
226         while(array[i + 1] != '\0'){
227             I2CMasterDataPut(I2C0_BASE, array[i++]);
228             I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);
229             while(I2CMasterBusy(I2C0_BASE));
230         }
231         I2CMasterDataPut(I2C0_BASE, array[i]);
232         I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
233         while(I2CMasterBusy(I2C0_BASE));
234     }
235 }
236 /**
237 * This function is used to receive 1 data on serial Data line
238 */
239 uint32_t I2CReceive(uint32_t slave_addr, uint8_t reg){
240     I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);
241     I2CMasterDataPut(I2C0_BASE, reg);
242     I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
243     while(I2CMasterBusy(I2C0_BASE));
244     I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, true);
245     I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
246     while(I2CMasterBusy(I2C0_BASE));
247     return I2CMasterDataGet(I2C0_BASE); //return data pulled from the specified register
248 }
249 /**
250 * This function is used to receive 2 data on serial Data line
251 */
252 uint16_t read2Bytes(uint8_t SLAVE_ADDRESS, uint8_t reg){
253     I2CMasterSlaveAddrSet(I2C0_BASE, SLAVE_ADDRESS, false);
254     I2CMasterDataPut(I2C0_BASE, reg);
255     I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
256     while (!(I2CMasterBusy(I2C0_BASE))) {};
257     while(I2CMasterBusy(I2C0_BASE)) {};
258     I2CMasterSlaveAddrSet(I2C0_BASE, SLAVE_ADDRESS, true);
259     I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START);
260     while (!(I2CMasterBusy(I2C0_BASE))) {};
261     while(I2CMasterBusy(I2C0_BASE)) {};
262     uint8_t data=I2CMasterDataGet (I2C0_BASE);
263     I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH);
264     while (!(I2CMasterBusy(I2C0_BASE))) {};
265     while(I2CMasterBusy(I2C0_BASE)) {};
266     uint8_t dummy=I2CMasterDataGet (I2C0_BASE);
267     return (((data<<8)+dummy)>>4);
268 }
269

```

6.11 Serial Communication

The Fire Bird V can communicate with other robots / devices serially using either wired link or wireless module. Serial communication is done in asynchronous mode. In the asynchronous mode, the common clock signal is not required at both the transmitter and receiver for data synchronization. As an example of serial communication code for interfacing Zigbee module is given below.

6.11.1 Connections

Pin	Plug and Play	uC based
RX	PB0(UART1)	PC6(UART3)
TX	PB1(UART1)	PC7(UART3)

6.11.2 Code for Plug and Play Board:

```

1 #include <stdlib.h>
2 #include <stdint.h>
3 #include <stdbool.h>
4 #include <math.h>
5 #include "inc/hw_memmap.h"
6 #include "inc/hw_types.h"
7 #include "driverlib/pin_map.h"
8 #include "driverlib/sysctl.h"
9 #include "driverlib/uart.h"
10 #include "driverlib/debug.h"
11 #include "driverlib/interrupt.h"
12 #include "driverlib/gpio.h"
13 #include "inc/tm4c123gh6pm.h"
14
15 void configCLK();
16 void peripheralEnable();
17 void uartEnable();
18 void uartInterruptEnable();
19 void UARTIntHandler(void);
20 void tranString(char *,char);
21 void uartInteger(int64_t number);
22
23 int main(){
24     configCLK();
25     peripheralEnable();
26     uartEnable();
27     uartInterruptEnable();
28     while(1){
29 }
30 }
31 /*****
32 * This function is used to setup Clock frequency of the controller
33 * It can be changed through codes
34 * In this we have set frequency as 40Mhz
35 * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
36 *****/
37 void configCLK(){
38     SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
39 }
40 /*****
41 * Enabling System Peripherals
42 * UART1 and PortB in this case
43 *****/
44 void peripheralEnable(){
45     SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
46     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
47 }
48 /*****
49 * This function is used to enable UART0
50 * The baudrate is set at 9600
51 *****/
52 void uartEnable(){
53     GPIOPinConfigure(GPIO_PB0_U1RX);
54     GPIOPinConfigure(GPIO_PB1_U1TX);
55     GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);
56     UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 9600,
57     (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
58 }
59 /*****
60 * This function is used to send a string of character with a delimiter
61 *****/
62 void tranString(char * data,char delimiter){
63     int k=0;
64     while(data[k]){
65         UARTCharPut(UART1_BASE,data[k++]);
66     }
67     UARTCharPut(UART1_BASE,delimiter);
68 }
69 /*****
70 * This function is used to enable UART1 Rx And Rx timeout Interrupt
71 *****/
72 void uartInterruptEnable(){
73     IntMasterEnable(); //Enable processor interrupt
74     IntEnable(INT_UART1); //Enable interrupt on UART0

```

```

75     UARTIntEnable(UART1_BASE, UART_INT_RX | UART_INT_RT); //Enable RX interrupt ant rx Timeout
76     interrupt
77     }
78     /**************************************************************************
79     * Interrupt sub routine for Uart0
80     * Change the name of ISR in startup file
81     * Refer to manual for proper instruction
82     *****/
83     void UARTIntHandler(void){
84         uint32_t ui32Status;
85         ui32Status = UARTIntStatus(UART1_BASE, true); //get interrupt status
86         UARTIntClear(UART1_BASE, ui32Status); //clear the asserted interrupts
87         while(UARTCharsAvail(UART1_BASE)){ //loop while there are chars
88             UARTCharPut(UART1_BASE, UARTCharGet(UART1_BASE));
89         }
90     }

```

6.11.3 Code for uC based Board:

```

1 #include <stdlib.h>
2 #include <stdint.h>
3 #include <stdbool.h>
4 #include <math.h>
5 #include "inc/hw_memmap.h"
6 #include "inc/hw_types.h"
7 #include "driverlib/pin_map.h"
8 #include "driverlib/sysctl.h"
9 #include "driverlib/uart.h"
10 #include "driverlib/debug.h"
11 #include "driverlib/interrupt.h"
12 #include "driverlib/gpio.h"
13 #include "inc/tm4c123gh6pm.h"
14
15 void configCLK();
16 void peripheralEnable();
17 void uartEnable();
18 void uartInterruptEnable();
19 void UARTIntHandler(void);
20 void transString(char *,char);
21 void uartInteger(int64_t number);
22
23 int main(){
24     configCLK();
25     peripheralEnable();
26     uartEnable();
27     uartInterruptEnable();
28     while(1){
29     }
30 }
31 void configCLK(){
32     SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
33 }
34 void peripheralEnable(){
35     SysCtlPeripheralEnable(SYSCTL_PERIPH_UART3);
36     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC); //Enablinig TIMER0
37 }
38 void uartEnable(){
39     GPIOPinConfigure(GPIO_PC6_U3RX); //Configure Pin PC6 as RX of U0
40     GPIOPinConfigure(GPIO_PC7_U3TX); //Configure Pin PC7 as TX of U0
41     GPIOPinTypeUART(GPIO_PORTC_BASE, GPIO_PIN_6 | GPIO_PIN_7);
42     UARTConfigSetExpClk(UART3_BASE, SysCtlClockGet(), 9600,
43     (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
44 }
45 void transString(char * data ,char delimiter){
46     int k=0;
47     while(data[k]){
48         UARTCharPut(UART3_BASE, data [k++]);
49     }
50     UARTCharPut(UART3_BASE, delimiter);
51 }
52 void uartInterruptEnable(){
53     IntMasterEnable(); //Enable processor interrupt
54     IntEnable(INT_UART3); //Enable interrupt on UART0
55     UARTIntEnable(UART3_BASE, UART_INT_RX | UART_INT_RT); //Enable RX interrupt ant rx Timeout
56     interrupt

```

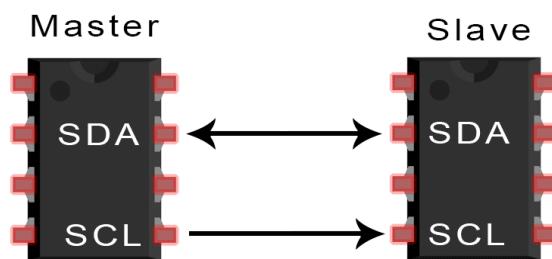
```

56
57     }
58     void UARTIntHandler(void){
59         uint32_t ui32Status;
60         ui32Status = UARTIntStatus(UART3_BASE, true); //get interrupt status
61         UARTIntClear(UART3_BASE, ui32Status); //clear the asserted interrupts
62         while(UARTCharsAvail(UART3_BASE)){ //loop while there are chars
63             UARTCharPut(UART3_BASE, UARTCharGet(UART3_BASE));
64         }
65     }
66 }
```

6.12 I2C Communication

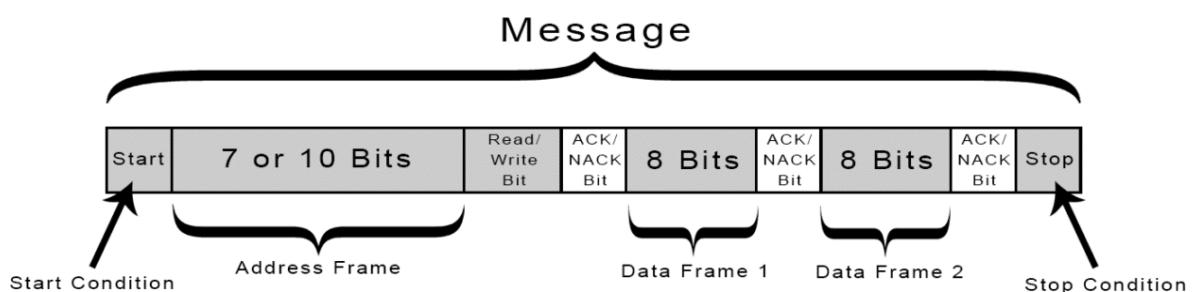
6.12.1 Introduction

I2C is a serial protocol for two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters, I/O interfaces and other similar peripherals in embedded systems. I2C combines the best features of SPI and UARTs. Like the Serial Peripheral Interface (SPI), it is only intended for short distance communications within a single device. Like Asynchronous Serial Interfaces (such as RS-232 or UARTs), it only requires two signal wires to exchange information.



6.12.2 Features

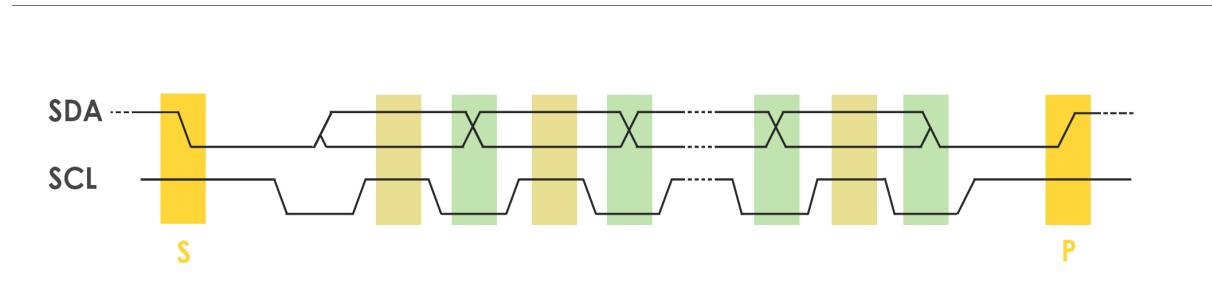
- 2 bidirectional lines SDA (serial data) and SCL (serial clock)
- Independent Master, Slave, and Monitor functions
- Supports both Multi-master and Multi-master with Slave functions
- One slave address can be selectively qualified with a bit mask or an address range in order to respond to multiple I2C bus addresses
- advanced features such as automatic multi master arbitration management



6.12.3 Working

In normal state both lines SDA and SCL are at high state this indicates the bus is free, any device can use this bus. Master will now initiate the transfer by triggering a start condition with slave address to which it want to communicate. The corresponding slave device matches its address with master sent address, if it matches then both starts communicating.

If logic low is sent for bit 0 then the master writes data to the slave device, otherwise next byte wise of the remaining data will be read from the slave device. After all the communication between the master and slave is over master will generate a stop condition indicating that communication is over and another slave can use the bus.



As an example of I2C communication code for interfacing port expander IC is given below. In this case MCP23017 I2C based IC is used. The data sheet of the IC can be downloaded here [MCP23017](#)

6.12.4 Connections

Pin	Plug and Play	uC based
SCL	PA6(I2C1)	PB2(I2C0)
SDA	PA7(I2C1)	PB3(I2C0)

6.12.5 Code for Plug and Play Board:

```

1 #include <stdarg.h>
2 #include <stdbool.h>
3 #include <stdint.h>
4 #include "inc/hw_i2c.h"
5 #include "inc/hw_memmap.h"
6 #include "inc/hw_types.h"
7 #include "inc/hw_gpio.h"
8 #include "driverlib/i2c.h"
9 #include "driverlib/sysctl.h"
10 #include "driverlib/gpio.h"
11 #include "driverlib/pin_map.h"
12 #include "inc/tm4c123gh6pm.h"
13 #include "driverlib/interrupt.h"
14 void setupCLK();
15 void peripheralEnable();
16 void gpioEnable();
17 void InitI2C1(void);
18 void I2CSendString(uint32_t slave_addr, char array[]);
19 void I2CSend(uint8_t slave_addr, uint8_t num_of_args, ...);
20 uint32_t I2CReceive(uint32_t slave_addr, uint8_t reg);
21 void portExpanderIO(unsigned char port, unsigned char pin);
22 void portExpanderSetOutput(unsigned char, unsigned char);
23 unsigned char portExpanderReadInput(unsigned char);
24 void portExpanderInterruptEnableAnyChange(unsigned char, unsigned char);
25 void portExpanderpullup(unsigned char, unsigned char);
26 void portExpanderInterruptHandler();
27 int main(void) {
28     setupCLK();
29     peripheralEnable();
30     gpioEnable();
31     InitI2C1();
32     portExpanderIO(0x00,0xff);
33     portExpanderIO(0x01,0x00);
34     portExpanderSetOutput(0x01,0x00);
35     portExpanderpullup(0x00,0x0f);
36     portExpanderInterruptEnableAnyChange(0x00,0xff);
37     while(1){
38     }
39 }
40 void setupCLK(){
41     SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);

```

```

42 }
43 void peripheralEnable(){
44 SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
45 }
46 void gpioEnable(){
47 GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_7);
48 GPIOPinTypeGPIOInput(GPIO_PORTC_BASE, GPIO_PIN_7);
49 GPIOPadConfigSet(GPIO_PORTC_BASE ,GPIO_PIN_7,GPIO_STRENGTH_2MA,GPIO_PIN_TYPE_STD_WPU);
50 }
51 void InitI2C1(void){
52 SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C1);
53 SysCtlPeripheralReset(SYSCTL_PERIPH_I2C1);
54 SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
55 GPIOPinConfigure(GPIO_PA6_I2C1SCL);
56 GPIOPinConfigure(GPIO_PA7_I2C1SDA);
57 GPIOPinTypeI2CSCL(GPIO_PORTA_BASE, GPIO_PIN_6);
58 GPIOPinTypeI2C(GPIO_PORTA_BASE, GPIO_PIN_7);
59 // Enable and initialize the I2C1 master module. Use the system clock for
60 // the I2C1 module. The last parameter sets the I2C data transfer rate.
61 // If false the data rate is set to 100kbps and if true the data rate will
62 // be set to 400kbps.
63 I2CMasterInitExpClk(I2C1_BASE, SysCtlClockGet(), false);
64 //clear I2C FIFOs
65 HWREG(I2C1_BASE + I2C_O_FIFOCCTL) = 80008000;
66 I2CSend(0x20,2,0xA,1<<6);
67 }
68 void I2CSend(uint8_t slave_addr, uint8_t num_of_args, ...){
69 // Tell the master module what address it will place on the bus when
70 // communicating with the slave.
71 I2CMasterSlaveAddrSet(I2C1_BASE, slave_addr, false);
72
73 //stores list of variable number of arguments
74 va_list vargs;
75
76 //specifies the va_list to "open" and the last fixed argument
77 //so vargs knows where to start looking
78 va_start(vargs, num_of_args);
79
80 //put data to be sent into FIFO
81 I2CMasterDataPut(I2C1_BASE, va_arg(vargs, uint32_t));
82
83 //if there is only one argument, we only need to use the
84 //single send I2C function
85 if(num_of_args == 1)
86 {
87 //Initiate send of data from the MCU
88 I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_SEND);
89
90 // Wait until MCU is done transferring.
91 while(I2CMasterBusy(I2C1_BASE));
92
93 // "close" variable argument list
94 va_end(vargs);
95 }
96
97 //otherwise, we start transmission of multiple bytes on the
98 //I2C bus
99 else
100 {
101 //Initiate send of data from the MCU
102 I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_START);
103
104 // Wait until MCU is done transferring.
105 while(I2CMasterBusy(I2C1_BASE));
106
107 //send num_of_args-2 pieces of data, using the
108 //BURST_SEND_CONT command of the I2C module
109 uint8_t i ;
110 for(i = 1; i < (num_of_args - 1); i++)
111 {
112 //put next piece of data into I2C FIFO
113 I2CMasterDataPut(I2C1_BASE, va_arg(vargs, uint32_t));
114 //send next data that was just placed into FIFO
115 I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);
116
117

```

```

118 // Wait until MCU is done transferring.
119 while(I2CMasterBusy(I2C1_BASE));
120 }
121
122 //put last piece of data into I2C FIFO
123 I2CMasterDataPut(I2C1_BASE, va_arg(vargs, uint32_t));
124 //send next data that was just placed into FIFO
125 I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
126 // Wait until MCU is done transferring.
127 while(I2CMasterBusy(I2C1_BASE));
128
129 //”close” variable args list
130 va_end(vargs);
131 }
132 }
133 //sends an array of data via I2C to the specified slave
134 void I2CSendString(uint32_t slave_addr, char array[])
135 {
136 // Tell the master module what address it will place on the bus when
137 // communicating with the slave.
138 I2CMasterSlaveAddrSet(I2C1_BASE, slave_addr, false);
139
140 //put data to be sent into FIFO
141 I2CMasterDataPut(I2C1_BASE, array[0]);
142
143 //if there is only one argument, we only need to use the
144 //single send I2C function
145 if(array[1] == '\0')
146 {
147 //Initiate send of data from the MCU
148 I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_SEND);
149
150 // Wait until MCU is done transferring.
151 while(I2CMasterBusy(I2C1_BASE));
152 }
153
154 //otherwise, we start transmission of multiple bytes on the
155 //I2C bus
156 else
157 {
158 //Initiate send of data from the MCU
159 I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_START);
160
161 // Wait until MCU is done transferring.
162 while(I2CMasterBusy(I2C1_BASE));
163
164 //initialize index into array
165 uint8_t i = 1;
166
167 //send num_of_args-2 pieces of data, using the
168 //BURST_SEND_CONT command of the I2C module
169 while(array[i + 1] != '\0')
170 {
171 //put next piece of data into I2C FIFO
172 I2CMasterDataPut(I2C1_BASE, array[i++]);
173
174 //send next data that was just placed into FIFO
175 I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);
176
177 // Wait until MCU is done transferring.
178 while(I2CMasterBusy(I2C1_BASE));
179 }
180
181 //put last piece of data into I2C FIFO
182 I2CMasterDataPut(I2C1_BASE, array[i]);
183
184 //send next data that was just placed into FIFO
185 I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
186
187 // Wait until MCU is done transferring.
188 while(I2CMasterBusy(I2C1_BASE));
189 }
190 }
191 //read specified register on slave device
192 uint32_t I2CReceive(uint32_t slave_addr, uint8_t reg)
193 {

```

```

194 //specify that we are writing (a register address) to the
195 //slave device
196 I2CMasterSlaveAddrSet(I2C1_BASE, slave_addr, false);

197 //specify register to be read
198 I2CMasterDataPut(I2C1_BASE, reg);

199 //send control byte and register address byte to slave device
200 I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_START);

201 //wait for MCU to finish transaction
202 while(I2CMasterBusy(I2C1_BASE));

203 //specify that we are going to read from slave device
204 I2CMasterSlaveAddrSet(I2C1_BASE, slave_addr, true);

205 //send control byte and read from the register we
206 //specified
207 I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);

208 //wait for MCU to finish transaction
209 while(I2CMasterBusy(I2C1_BASE));

210 //return data pulled from the specified register
211 return I2CMasterDataGet(I2C1_BASE);
212 }

213 void portExpanderIO(unsigned char port, unsigned char pin){
214 I2CSend(0x20, 2, port, pin);
215 }

216 void portExpanderSetOutput(unsigned char port, unsigned char pin){
217 I2CSend(0x20, 2, port+(0x12), pin);
218 }

219 unsigned char portExpanderReadInput(unsigned char port){
220 return (I2CReceive(0x20, (port+12)));
221 }

222 void portExpanderInterruptEnableAnyChange(unsigned char port, unsigned char pin){
223 portExpanderIO(port, pin);
224 I2CSend(0x20, 2, (0x04)+port, pin);
225 I2CSend(0x20, 2, (0x08)+port, pin);
226 GPIOIntDisable(GPIO_PORTC_BASE, GPIO_PIN_7); // Disable interrupt for PF4 (in case it was
227 enabled)
228 GPIOIntClear(GPIO_PORTC_BASE, GPIO_PIN_7); // Clear pending interrupts for PF4
229 GPIOIntRegister(GPIO_PORTC_BASE, portExpanderInterruptHandler); // Register our handler
230 }

231 function for port F
232     GPIOIntTypeSet(GPIO_PORTC_BASE, GPIO_PIN_7, GPIO_FALLING_EDGE); // Configure PF4 for
233 falling edge trigger
234     GPIOIntEnable(GPIO_PORTC_BASE, GPIO_PIN_7);
235 }

236 void portExpanderInterruptHandler(){
237     if(GPIOIntStatus(GPIO_PORTC_BASE, false)&GPIO_PIN_7){
238         if(I2CReceive(0x20, 0x0e)&0x01==0x01){
239             portExpanderSetOutput(0x01, 0xff);
240         }
241         I2CReceive(0x20, 0x10);
242         I2CReceive(0x20, 0x11);
243         GPIOIntClear(GPIO_PORTC_BASE, GPIO_PIN_7);
244     }
245 }

246 void portExpanderpullup(unsigned char port, unsigned char pin){
247 I2CSend(0x20, 2, (0x0C)+port, pin);
248 }

249

```

6.12.6 Code for uC based Board:

```

1 #include <stdarg.h>
2 #include <stdbool.h>
3 #include <stdint.h>
4 #include "inc/hw_i2c.h"
5 #include "inc/hw_memmap.h"
6 #include "inc/hw_types.h"
7 #include "inc/hw_gpio.h"
8 #include "driverlib/i2c.h"
9 #include "driverlib/sysctl.h"
10 #include "driverlib/gpio.h"
11 #include "driverlib/pin_map.h"

```

```

12 #include "inc/tm4c123gh6pm.h"
13 #include "driverlib/interrupt.h"
14
15 void setupCLK();
16 void peripheralEnable();
17 void gpioEnable();
18 void InitI2C1(void);
19 void I2CSendString(uint32_t slave_addr, char array []);
20 void I2CSend(uint8_t slave_addr, uint8_t num_of_args, ...);
21 uint32_t I2CReceive(uint32_t slave_addr, uint8_t reg);
22 void portExpanderIO(unsigned char port, unsigned char pin);
23 void portExpanderSetOutput(unsigned char, unsigned char);
24 unsigned char portExpanderReadInput(unsigned char);
25 void portExpanderInterruptEnableAnyChange(unsigned char, unsigned char);
26 void portExpanderpullup(unsigned char, unsigned char);
27 void portExpanderInterruptHandler();
28
29 int main(void) {
30     setupCLK();
31     peripheralEnable();
32     gpioEnable();
33     InitI2C1();
34     portExpanderIO(0x00,0xff);
35     portExpanderIO(0x01,0x00);
36     portExpanderSetOutput(0x01,0x00);
37     portExpanderpullup(0x00,0x0f);
38     portExpanderInterruptEnableAnyChange(0x00,0xff);
39     while(1){
40 }
41 }
42 ****
43 * This function is used to setup Clock frequency of the controller
44 * It can be changed through codes
45 * In this we have set frequency as 40Mhz
46 * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
47 ****
48 void setupCLK(){
49     SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
50 }
51 ****
52 * Enabling System Peripherals
53 * PortC in this case
54 ****
55 void peripheralEnable(){
56     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
57 }
58 ****
59 * Making Pin Port C7 as input with weak pull up
60 ****
61 void gpioEnable(){
62     GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_7);
63     GPIOPinTypeGPIOInput(GPIO_PORTC_BASE, GPIO_PIN_7);
64     GPIOPadConfigSet(GPIO_PORTC_BASE ,GPIO_PIN_7, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
65 }
66 ****
67 * This function is used to enable I2C1
68 ****
69 void InitI2C1(void){
70     SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C1);
71     SysCtlPeripheralReset(SYSCTL_PERIPH_I2C1);
72     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
73     GPIOPinConfigure(GPIO_PA6_I2C1SCL);
74     GPIOPinConfigure(GPIO_PA7_I2C1SDA);
75     GPIOPinTypeI2CSCL(GPIO_PORTA_BASE, GPIO_PIN_6);
76     GPIOPinTypeI2C(GPIO_PORTA_BASE, GPIO_PIN_7);
77     // Enable and initialize the I2C1 master module. Use the system clock for
78     // the I2C1 module. The last parameter sets the I2C data transfer rate.
79     // If false the data rate is set to 100kbps and if true the data rate will
80     // be set to 400kbps.
81     I2CMasterInitExpClk(I2C1_BASE, SysCtlClockGet(), false);
82     //clear I2C FIFOs
83     HWREG(I2C1_BASE + I2C_O_FIFOCCTL) = 80008000;
84     I2CSend(0x20,2,0xA,1<<6);
85 }
86 ****
87 * This function is used to send n number of data on serial Data line

```

```

*****void I2CSend(uint8_t slave_addr , uint8_t num_of_args , ...){
    I2CMasterSlaveAddrSet(I2C1_BASE , slave_addr , false);
    va_list vargs;
    va_start(vargs , num_of_args);
    I2CMasterDataPut(I2C1_BASE , va_arg(vargs , uint32_t));
    if(num_of_args == 1){
        I2CMasterControl(I2C1_BASE , I2C_MASTER_CMD_SINGLE_SEND);
        while(I2CMasterBusy(I2C1_BASE));
        va_end(vargs);
    }
    else{
        I2CMasterControl(I2C1_BASE , I2C_MASTER_CMD_BURST_SEND_START);
        while(I2CMasterBusy(I2C1_BASE));
        uint8_t i ;
        for(i = 1; i < (num_of_args - 1); i++){
            I2CMasterDataPut(I2C1_BASE , va_arg(vargs , uint32_t));
            I2CMasterControl(I2C1_BASE , I2C_MASTER_CMD_BURST_SEND_CONT);
            while(I2CMasterBusy(I2C1_BASE));
        }
        I2CMasterDataPut(I2C1_BASE , va_arg(vargs , uint32_t));
        I2CMasterControl(I2C1_BASE , I2C_MASTER_CMD_BURST_SEND_FINISH);
        while(I2CMasterBusy(I2C1_BASE));
        va_end(vargs);
    }
}
*****/
* This function is used to send string data on serial Data line
*****
void I2CSendString(uint32_t slave_addr , char array[]){
    I2CMasterSlaveAddrSet(I2C1_BASE , slave_addr , false);
    I2CMasterDataPut(I2C1_BASE , array[0]);
    if(array[1] == '\0'){
        I2CMasterControl(I2C1_BASE , I2C_MASTER_CMD_SINGLE_SEND);
        while(I2CMasterBusy(I2C1_BASE));
    }
    else{
        I2CMasterControl(I2C1_BASE , I2C_MASTER_CMD_BURST_SEND_START);
        while(I2CMasterBusy(I2C1_BASE));
        uint8_t i = 1;
        while(array[i + 1] != '\0'){
            I2CMasterDataPut(I2C1_BASE , array[i++]);
            I2CMasterControl(I2C1_BASE , I2C_MASTER_CMD_BURST_SEND_CONT);
            while(I2CMasterBusy(I2C1_BASE));
        }
        I2CMasterDataPut(I2C1_BASE , array[i]);
        I2CMasterControl(I2C1_BASE , I2C_MASTER_CMD_BURST_SEND_FINISH);
        while(I2CMasterBusy(I2C1_BASE));
    }
}
*****/
* This function is used to receive 1 data on serial Data line
*****
uint32_t I2CReceive(uint32_t slave_addr , uint8_t reg){
    I2CMasterSlaveAddrSet(I2C1_BASE , slave_addr , false);
    I2CMasterDataPut(I2C1_BASE , reg);
    I2CMasterControl(I2C1_BASE , I2C_MASTER_CMD_BURST_SEND_START);
    while(I2CMasterBusy(I2C1_BASE));
    I2CMasterSlaveAddrSet(I2C1_BASE , slave_addr , true);
    I2CMasterControl(I2C1_BASE , I2C_MASTER_CMD_SINGLE_RECEIVE);
    while(I2CMasterBusy(I2C1_BASE));
    return I2CMasterDataGet(I2C1_BASE);
}
*****/
* This function is used to set PORTA or PORTB pins as I/O
*****
void portExpanderIO(unsigned char port , unsigned char pin){
    I2CSend(0x20,2,port,pin);
}
*****/
* This function is used to give PORTA or PORTB pin output values
*****
void portExpanderSetOutput(unsigned char port , unsigned char pin){
    I2CSend(0x20,2,port+(0x12),pin);
}
*****/

```

```

164 * This function is used to read PORTA or PORTB pin values
165 ****
166 unsigned char portExpanderReadInput(unsigned char port){
167     return(I2CReceive(0x20,(port+12)));
168 }
169 ****
170 * This function is used to enable interrupt on PORTA or PORTB pin
171 ****
172 void portExpanderInterruptEnableAnyChange(unsigned char port,unsigned char pin){
173     portExpanderIO(port,pin);
174     I2CSend(0x20,2,(0x04)+port,pin);
175     I2CSend(0x20,2,(0x08)+port,pin);
176     GPIOIntDisable(GPIO_PORTC_BASE,GPIO_PIN_7);           // Disable interrupt for PF4 (in case it
was enabled)
177     GPIOIntClear(GPIO_PORTC_BASE,GPIO_PIN_7);           // Clear pending interrupts for PF4
178     GPIOIntRegister(GPIO_PORTC_BASE,portExpanderInterruptHandler); // Register our handler
function for port F
179     GPIOIntTypeSet(GPIO_PORTC_BASE,GPIO_PIN_7,GPIO_FALLING_EDGE); // Configure PF4
for falling edge trigger
180     GPIOIntEnable(GPIO_PORTC_BASE,GPIO_PIN_7);
181 }
182 ****
183 * Pin change interrupt service routine
184 ****
185 void portExpanderInterruptHandler(){
186     if(GPIOIntStatus(GPIO_PORTC_BASE, false)&GPIO_PIN_7){
187         if(I2CReceive(0x20,0x0e)&0x01==0x01){
188             portExpanderSetOutput(0x01,0xff);
189         }
190         I2CReceive(0x20,0x10);
191         I2CReceive(0x20,0x11);
192         GPIOIntClear(GPIO_PORTC_BASE,GPIO_PIN_7);
193     }
194 }
195 ****
196 * This function is used to give external 100K pull toPORTA or PORTB pin
197 ****
198 void portExpanderpullup(unsigned char port,unsigned char pin){
199     I2CSend(0x20,2,(0x0C)+port,pin);
200 }
201

```